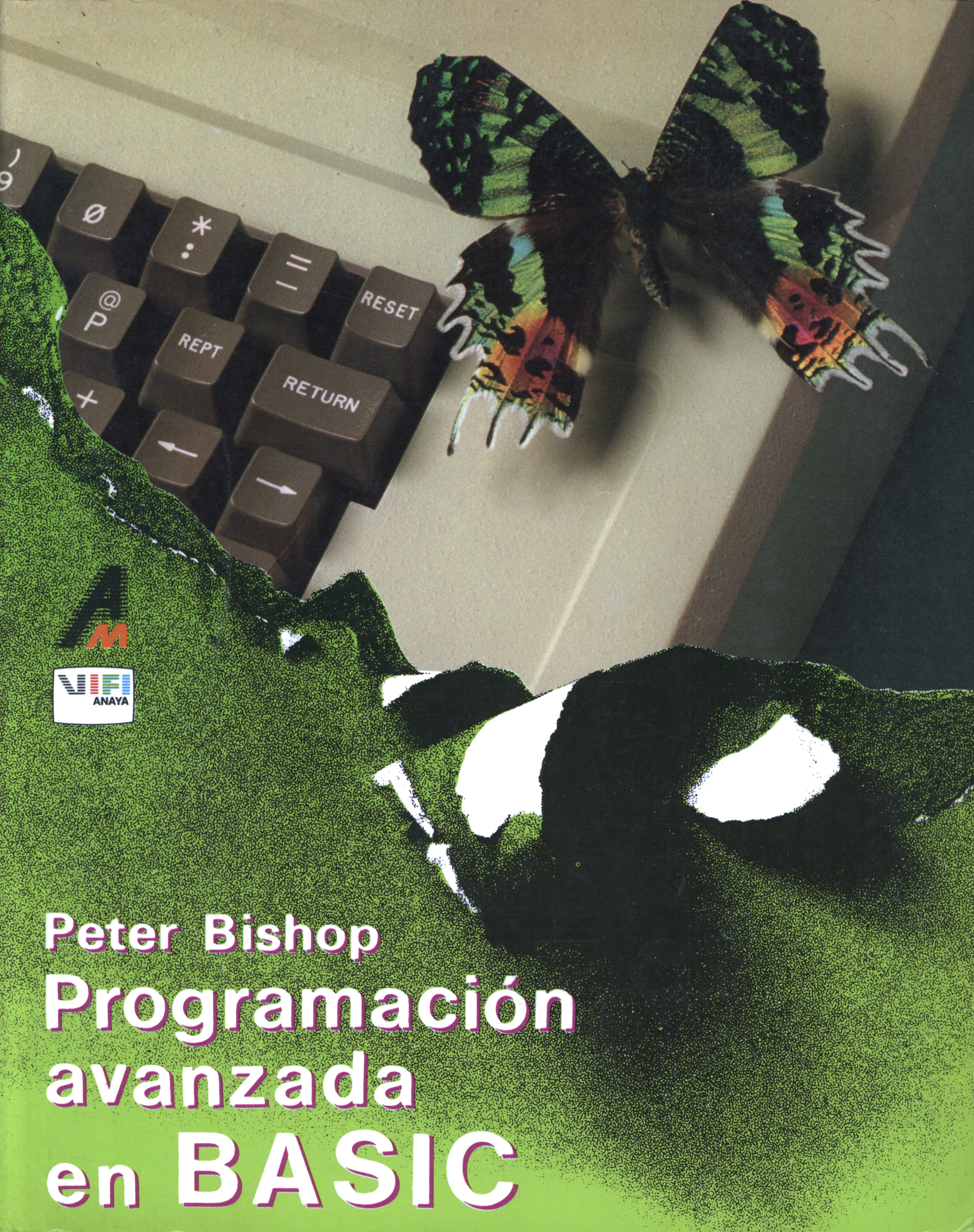

RESET
REPT
RETURN
AM
VIFI
ANAYA
Peter Bishop
Programación
avanzada
en BASIC

PROGRAMACION AVANZADA EN BASIC

Programación avanzada en BASIC

Peter Bishop



ANAYA MULTIMEDIA

MICROINFORMATICA

Título de la obra original:
«FURTHER COMPUTER PROGRAMMING IN BASIC»

Traducción de: Alfredo Cruz

Diseño de colección: Antonio Lax.

Publicado por primera vez por
Thomas Nelson and Sons Ltd., 1982

Villafranca, 22. 28028 Madrid
Depósito legal: S. 76-1985
ISBN: 84-7614-013-4
Printed in Spain
Imprime: Gráficas Ortega, S.A.
Polígono El Montalvo - Salamanca, 1985

Contenido

Indice

Acercamiento a la redacción de programas

Prólogo

La finalidad de este libro es enseñar a programar en un lenguaje de alto nivel, y utiliza el Basic como medio para conseguirlo. En lo que se insiste es en los conceptos, en los conocimientos que es preciso adquirir y en las técnicas de programación y en la necesidad de enfocar la labor de diseñar y redactar un programa de forma disciplinada. El texto está adaptado a las necesidades de todos los cursos de informática y ordenadores de nivel superior que se imparten en el Reino Unido. Se incluyen preguntas realizadas en exámenes de estos cursos, así como una lista de ideas para proyectos a realizar, que generalmente se incluyen en dichos cursos.

También se han tenido en cuenta al reescribir el libro las necesidades de la industria informática, sobre todo por lo que respecta a las normas de programación y de escritura de programas.

Pero un libro como éste satisface un abanico de necesidades más amplio: así, es interesante para los profesores que estén preparando cursos de niveles básico o avanzado, para alumnos que sigan cursos de ampliación o para los que piensen estudiar informática en la universidad. Igualmente es idóneo para el poseedor de un microordenador que no se conforme con sacarle un partido meramente superficial.

El libro está organizado en cinco partes, formadas cada una de ellas por varios capítulos. En la primera parte se exponen los elementos principales del Basic, subrayando al mismo tiempo los conceptos

fundamentales de programación. Se han evitado deliberadamente los «añadidos» al lenguaje que sólo son compatibles con marcas determinadas de ordenadores. Los diagramas de flujo se usan para ilustrar la estructura de los programas, pero no se consideran como parte esencial del diseño de los mismos. Los programas propuestos como ejemplos y los ejercicios de esta primera parte son siempre sencillos y breves. La segunda parte está al margen de cualquier lenguaje de programación y se refiere a la forma de escribir programas. Aborda cuestiones vitales de diseño, estructura, verificación, documentación y mantenimiento de programas. Los programas en Basic usados como ejemplos sirven para ilustrar las técnicas expuestas.

Las partes tercera y cuarta relacionan la programación con la teoría general del trabajo con ordenadores. La tercera describe tres operaciones fundamentales de programación —clasificar, buscar y refundir *(merge)*— y la cuarta se dedica al estudio de pilas, colas, listas y otras estructuras de datos.

La última parte se reserva a las aplicaciones de la programación. Estos capítulos son autónomos y examinan temas como la realización de gráficos, la simulación o el análisis sintáctico. Aunque a lo largo del texto se utiliza un nivel de conocimientos matemáticos muy bajo, en esta última parte hay dos capítulos que constituyen una excepción a ese respecto, y que permiten a los que dispongan de algunos conocimientos de análisis numérico combinarlos con el estudio de la programación; no obstante, pueden omitirse sin pérdida de continuidad.

El texto termina con una serie de propuestas de trabajos, un ejercicio de repaso, un resumen de instrucciones y funciones del Basic y un glosario de términos técnicos.

Agradecimientos

En la planificación y escritura de este libro he contado con la colaboración de varias personas. Valerie Downes, del Imperial College, revisó el plan general y bastantes capítulos. John Darlington me dio algunos consejos sobre diseño de programas. Y Jill Rout mecanografió los capítulos de muestra.

También quiero que conste mi gratitud para con Patrick Sutton, que revisó el texto, y Ruth Bush, que mecanografió el manuscrito completo.

También deseo manifestar mi agradecimiento al Tribunal de Exámenes del Certificado de Educación General, a a la Junta de Matriculaciones, a la delegación de Oxford para exámenes locales y al Departamento de Exámenes de la Universidad de Londres, por su amable permiso de reproducción de algunos exámenes pasados.


RESET
RETURN

1
Introducción

La finalidad de este libro es proporcionar una base amplia y profunda de programación de ordenadores en un lenguaje de alto nivel, y para ello se empleará como medio el lenguaje Basic. Pero el énfasis se centra en los conceptos, las facultades y las técnicas de programación más que en los detalles del lenguaje.

Aunque el texto está pensado para cursos avanzados de programación, no presupone conocimientos anteriores de esta disciplina ni del lenguaje Basic; salvo un par de capítulos especializados situados hacia el final del volumen, el resto del libro no exige más conocimientos matemáticos que los de sentido común. A quienes tengan ya alguna experiencia en programación, este texto aportará una visión renovadora y considerablemente más profunda de lo que suele ser habitual.

Estas lecciones de programación tratarán también de desarrollar una serie de facultades, y en particular:

- el acercamiento sistemático a la técnica de programación;
- la capacidad para analizar un trabajo en términos de conceptos, técnicas de programación y de dispositivos de cálculo disponibles;
- el talento para diseñar y escribir programas de calidad elevada;
- el conocimiento de los conceptos de la estructura de programas y de la forma de aplicarlos;

- la familiarización, por medio de la programación, con una serie de conceptos fundamentales del trabajo con ordenadores, como las estructuras de datos, y de operaciones, como la ordenación y refundición;
- la capacidad para comprender programas escritos por otras personas;
- la capacidad para realizar pruebas sistemáticas de un programa y para identificar y corregir errores;
- la capacidad para escribir descripciones correctas del funcionamiento de un programa.

La programación supone el empleo de otras facultades de tipo más general, sobre todo de la capacidad de pensar con claridad y de la habilidad para escribir en un español conciso y comprensible.

Quizá estos objetivos parezcan ambiciosos y un tanto intimidatorios, pero las ventajas de poseer las facultades descritas son considerables y compensan sobradamente el esfuerzo invertido en desarrollarlas y conservarlas.

La programación ocupa una posición límite entre el arte y la ciencia. Es una labor creativa, pero a la vez está limitada por una serie de normas y convenciones. Disciplina es probablemente el término que mejor describe la técnica de la programación: aprender a programar es adquirir una **disciplina** mental.

1.1 Naturaleza de los programas de ordenador

Antes de empezar a presentar técnicas de programación en un lenguaje determinado, es necesario retroceder un poco y examinar la programación en perspectiva, un ejercicio tan importante para los noveles como para los que ya tienen alguna experiencia.

El resto de este capítulo está destinado a clarificar algunas ideas sobre la naturaleza de los programas de ordenador y a establecer el entorno en que operan. Este material guarda diferentes relaciones con el resto del libro; en primer lugar, varias de las ideas generales expuestas aquí se desarrollarán más adelante de manera más específica. Este capítulo aporta a gran parte de los demás una idea de dirección y finalidad. El dedicado al diseño de programas, situado hacia la mitad del libro, constituye una ampliación bastante rigurosa de las ideas presentadas en este que nos ocupa.

1.2

Formas de considerar los programas

Examinaremos a continuación tres formas diferentes de considerar un programa de ordenador: la visión elemental, la visión instrumental y la visión de envoltura o capa. La exposición de estos puntos de vista permitirá entrever la respuesta a una pregunta bastante escurridiza: ¿qué es un programa de ordenador?

1.3

La visión elemental

Este punto de vista considera que un programa es una serie de instrucciones que se dan a un ordenador. Algo así como decir que una casa es un montón ordenado de ladrillos: es una afirmación correcta, pero insuficiente a muchos efectos. De la misma manera que una casa es mucho más que un conjunto de ladrillos, un programa es mucho más que una serie de instrucciones.

La consideración del programa como una solución a un problema concreto constituye una extensión del mismo punto de vista y puede llevar a conclusiones erróneas. Esta postura procede de la época en que los programas se usaban para resolver problemas matemáticos; tales programas eran de finalidad única y se tiraban una vez resuelto el problema. En la práctica actual, esa clase de programas son extraordinariamente escasos.

1.4

La visión instrumental

Esta forma de ver los programas deriva de una forma particular de ver los ordenadores. Un ordenador es una máquina de tipo general que sirve para procesar información y, por tanto, un instrumento capaz de realizar operaciones muy variadas. En este contexto, el programa serviría para "centrar" al ordenador en la realización de una operación perfectamente definida. En otras palabras: el programa genera un tipo específico de máquina que a continuación se utiliza de una forma determinada. De este punto de vista se derivan importantes consecuencias que se discutirán a continuación.

Una herramienta no sirve de gran cosa si se rompe con frecuencia. En consecuencia, los programas deben ser capaces de funcionar mu-

chas veces, ser resistentes y mostrarse capaces de soportar el uso indebido, accidental o deliberado.

La herramienta generada por un programa ha de ser compatible con su medio ambiente de trabajo. En otras palabras: los programas deben estar orientados hacia las necesidades del usuario, ser lo más sencillos de usar que sea posible y coordinarse con otros posibles trabajos desarrollados por el usuario. En la práctica, la mayor parte de las tareas realizadas por los programas forman parte de operaciones más generales, y deben escribirse teniendo en cuenta las necesidades planteadas por esas operaciones más amplias.

Todas estas exigencias se suman a la necesidad de que el programa esté bien diseñado y tenga una estructura fiable. La cuestión del diseño de programas se aborda con más detalle en el capítulo 11, pero debe tenerse en cuenta desde el principio.

1.5 La visión multicapa

Otra alternativa es considerar a los programas como capas situadas entre el soporte físico *(hardware)* del ordenador y el mundo exterior. En este contexto, los programas reciben con frecuencia la denominación de **logical** *(software)*. Un ordenador funcional constaría de un núcleo físico rodeado por una capa de logical (véase figura 1.1).

Algunos programas convierten a un ordenador en una máquina capaz de trabajar con otros programas, de manera que el soporte físico aparecerá rodeado por una envoltura lógica de varias capas. La capa más interna interacciona directamente con el soporte físico y la

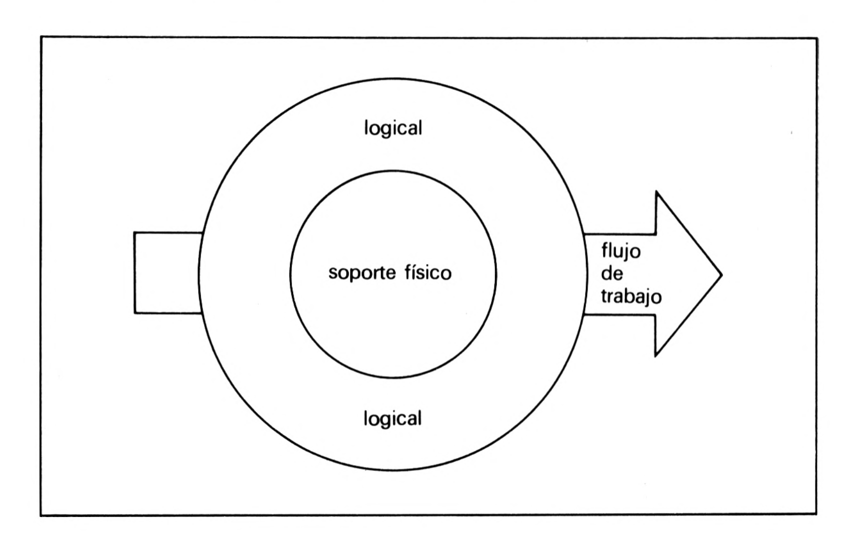


Figura 1.1
Soportes físico y logical

más externa con el usuario; se dice, además, que cada una de las capas **soporta** a las que le son externas.

Una disposición de logical multicapa muy común es la siguiente:

1. la capa interna es el **sistema operativo,** que interacciona directamente con el soporte físico y administra los recursos del ordenador;
2. a continuación viene la capa de los **programas traductores,** que traducen un lenguaje de alto nivel, como el Basic, al lenguaje máquina utilizado por el ordenador;
3. la capa más externa está constituida por los **programas de aplicaciones,** que colocan al ordenador en disposición de realizar tareas determinadas.

Esta disposición aparece ilustrada en la figura 1.2. En este libro examinaremos programas pertenecientes a las tres capas que acaban de referirse.

Relacionada con esta forma de considerar los programas está la idea de **interfaz.** Se llama así al espacio de contacto entre un elemento de un sistema informático (físico o lógico) y otro. Todos los programas tienen interfaces, que constituyen sus puntos de contacto con el soporte físico o con otras capas del logical o con el usuario. Así, una aplicación escrita en Basic tiene un interfaz con el usuario y otro con el programa traductor del lenguaje Basic, y dentro de ese mismo programa puede haber diferentes partes conectadas entre sí por medio de otros interfaces.

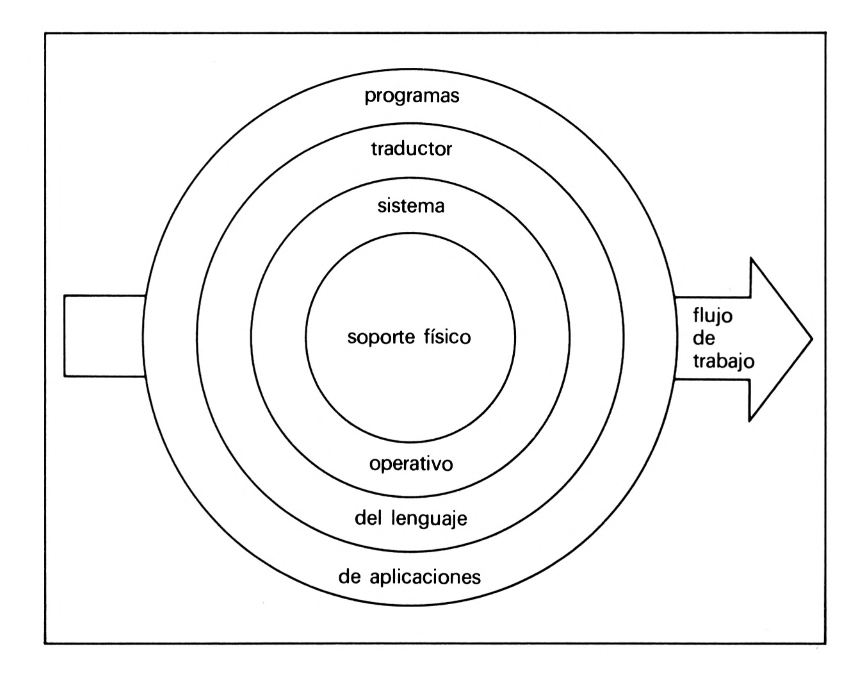


Figura 1.2
Niveles del logical

Otra idea que ineludiblemente debe introducirse en este punto es la de **algoritmo.** Un algoritmo es una descripción escrita en lenguaje sencillo de los pasos necesarios para desarrollar una tarea. La redacción del algoritmo correspondiente a la tarea que ha de llevar a cabo un programa constituye una parte esencial del diseño de programas.

1.6 Conclusión

En este capítulo se han bosquejado los objetivos generales del libro y se ha realizado una breve incursión en la naturaleza de los programas. Varias de las ideas presentadas son esenciales para comprender el resto del libro.

Los puntos más importantes pueden resumirse como sigue:

- La programación es a la par un arte y una ciencia, y disciplina es la palabra que mejor describe esta actividad.
- Un programa es algo más que un simple conjunto de instrucciones.
- Considerar un programa como una solución a un problema determinado es una actitud anticuada que, a veces, puede inducir a error.
- Con frecuencia es muy útil considerar un programa como algo que centra una máquina de tipo general (un ordenador) en la realización de una tarea específica.
- Un programa debe diseñarse de manera que pueda utilizarse muchas veces, incluso en circunstancias adversas.
- Los programas, o logical, pueden considerarse como capas dispuestas entre el soporte físico de un ordenador y el mundo exterior. Cada una de las capas dispone de interfaces por medio de los que interacciona con las demás.

1 Ejercicio

1. Defina brevemente los siguientes términos: disciplina, logical, interfaz, algoritmo.

2. *a*) Resuma en una o dos frases cada una de las tres formas de considerar los programas expuestas en este capítulo.
   b) Comente los tres puntos de vista.

3. En términos de las ideas expuestas en este capítulo, ¿qué tienen en común la programación y la composición tipográfica?

4. Si ya ha escrito programas de ordenador, valórelos críticamente y determine cuántos son de finalidad única, es decir, cuántos están diseñados para ser utilizados una sola vez y descartados a continuación. Explique la respuesta.


RESET
RETURN

2
El lenguaje Basic

En este capítulo se esboza el desarrollo del lenguaje Basic y se examina su función en la programación tal como se practica en la actualidad. Estudia el problema planteado por las numerosas versiones de ese lenguaje que ahora están en uso y expone la solución utilizada en este libro. El capítulo termina con la discusión de algunos de los conceptos fundamentales del lenguaje Basic.

El material de este capítulo constituye la base de lo que se construirá a lo largo del resto del libro. Antes de seguir adelante es imprescindible esforzarse por entender claramente las pocas ideas clave presentadas en esta parte del texto.

2.1 Desarrollo del lenguaje Basic

El lenguaje Basic fue desarrollado en Dartmouth College, Estados Unidos, por Thomas E. Kurtz y John Kemeny entre 1963 y 1964. El nombre Basic está formado por las iniciales de **B**eginer's **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode (código simbólico de instrucciones de tipo general para principiantes). La finalidad del nuevo lenguaje era iniciar a los estudiantes en la programación de ordenadores.

Desde su introducción el Basic se ha difundido muchísimo, y en la actualidad es el más popular de los lenguajes de introducción a la programación. Se ha implementado en todos los tipos de ordenador imaginables, desde los micros más elementales hasta las máquinas más potentes. Sus aplicaciones han pasado de la educación a la industria y el comercio, campos en los que ahora ocupa una posición destacada. Ocupa, además, el tercer lugar en cuanto a popularidad general, detrás del Cobol y el Fortran.

2.2 Objetivos del lenguaje Basic

El Basic se creó con la idea de que fuese fácil de aprender y sencillo de usar. Es un lenguaje de tipo general, apropiado para gran diversidad de aplicaciones. Por último, se muestra tan potente en la manipulación de números como en la de caracteres alfabéticos.

El Basic es un lenguaje **interactivo,** es decir, que facilita el diseño de programas que permiten mantener "conversaciones" entre el usuario y el ordenador. La capacidad de creación de gráficos de las versiones más recientes ha potenciado esta característica.

2.3 El problema: demasiadas versiones del Basic

Como el Basic se ejecuta cada vez más en ordenadores, se ha visto contaminado por un número creciente de variaciones. La publicación en 1978 de las versiones de referencia del Basic norteamericana y europea no ha podido hacer gran cosa por neutralizar esa tendencia.

La situación se explica por varias razones. En primer lugar, el deseo de cada uno de los fabricantes de ordenadores de lanzar productos superiores a los de la competencia, lo que les ha llevado a ciertas ampliaciones, como el tratamiento de matrices o el desarrollo de posibilidades de creación de gráficos muy perfeccionadas. Por otra parte, los microordenadores muy pequeños deben condensar el sistema de traducción del Basic en una memoria muy reducida, lo que conduce a la aparición de versiones abreviadas del lenguaje. Se trata de simples subconjuntos del lenguaje mismo que, desgraciadamente, tienen en algunos casos características muy infrecuentes.

En consecuencia, es casi imposible transferir un programa escrito en Basic de un ordenador a otro sin hacer previamente algunas modificaciones. Estos problemas están apartando al Basic de su función original de lenguaje de enseñanza sencillo.

2.4

La solución: un lenguaje de referencia y muchas implementaciones

El problema de un programa de ordenador con diferentes versiones no es nuevo, y surgió por vez primera durante el desarrollo del Algol, en 1958. La solución adoptada entonces es la misma adoptada aquí: un lenguaje de **referencia** y numerosas **implementaciones.**

2.5

El lenguaje de referencia

La versión de referencia del Basic funciona con un mínimo de modificaciones en cualquier ordenador que soporte dicho lenguaje. Contiene todos los elementos esenciales del Basic, pero es independiente de cualquier ordenador en particular. Está diseñado para ilustrar la estructura lógica del programa y para poder leerse con facilidad. No tiene por objetivo ser lo más breve posible ni lo más eficaz posible en un ordenador determinado.

El lenguaje de referencia utilizado en este texto procede de dos fuentes: Basic Microsoft, utilizado en muchos microordenadores; BASICA, utilizado en los microordenadores IBM PC. En general, el lenguaje de referencia contiene características comunes a las versiones mencionadas.

Salvo indicación en contrario, todos los programas de este libro están escritos en la versión de referencia, y es preciso insistir que en algunos casos será preciso modificarlos para poder utilizarlos en un ordenador concreto. De todas formas, las modificaciones necesarias son siempre reducidas, fáciles de realizar y, lo más importante, nunca afectarán a la estructura lógica del programa.

La limitación al lenguaje de referencia obliga a prescindir en este libro de algunas características muy potentes disponibles en ciertas máquinas, como la posibilidad de crear gráficos, los relojes en tiempo real o las interrupciones programables por el usuario. Sin embargo, la finalidad del texto es apoyar en una base firme los conceptos y técnicas esenciales de la programación, lo que permitirá al futuro programador sacar el máximo partido a dichas opciones si en algún momento las tiene a su disposición.

2.6

Implementación del lenguaje

Se llama implementación del lenguaje a cualquier versión del Basic compatible con un ordenador concreto. Algunos modelos son capaces de trabajar con varias versiones; los detalles sobre tales lenguajes figuran únicamente en los manuales de instrucciones de los ordenadores.

En la mayor parte de los casos, la traducción de los programas escritos en el lenguaje de referencia al de ejecución práctica adecuado se deja como ejercicio para el lector. El principio fundamental es conseguir que el programa funcione sin alterar su estructura lógica, aunque en algunos casos —pocos— sea imprescindible alterar dicha estructura o exista la posibilidad de simplificar la redacción del programa. En tales casos se incluyen ejemplos en una implementación.

2.7

Elementos esenciales del lenguaje Basic

El resto de este capítulo irá dedicado a la presentación de algunos elementos esenciales del lenguaje Basic, elementos que se refieren a la estructura general del programa, a la estructura de cada línea del programa, a las formas de representación de los datos y a las observaciones.

2.8

Estructura del programa

Un programa Basic consiste en una serie de **instrucciones,** conocidas por lo general como sentencias. El programa puede contener uno o más **subprogramas,** a los que se llama desde un **programa principal;** los subprogramas se tratarán en el capítulo 9. La estructura lógica del programa se trata a fondo en muchos sitios del texto, pero sobre todo en el capítulo 11.

2.9

Estructura de la línea

En la versión de referencia del Basic, cada sentencia ocupa una línea completa; la línea empieza siempre por un **número de línea,** al que sigue una **palabra de instrucción.**

Los números de línea son enteros positivos inferiores a cierto límite que depende de la implementación. Las líneas de programa pueden cargarse en el ordenador en cualquier orden, pero siempre se ejecutan en el orden de numeración y con arreglo a la posible presencia de ramificaciones y bucles (estos elementos se estudiarán en los capítulos 4 y 5, respectivamente). Es práctica común numerar las líneas con múltiplos de 5 o de 10, porque así es fácil insertar líneas nuevas posteriormente. La última línea introducida reemplaza a cualquiera anterior identificada mediante el mismo número.

Las palabras de instrucción determinan el tipo de operación que ha de emprenderse. El lenguaje Basic tiene alrededor de una veintena de esas palabras, que se presentarán a lo largo de los nueve primeros capítulos del texto.

Veamos a continuación algunos ejemplos de líneas de un programa escrito en Basic:

```
10 REM UN PROGRAMA EN BASIC ESTA FORMADO POR
15 REM UNA SERIE DE LINEAS. CADA UNA DE ELLAS
20 REM COMIENZA CON UN
25 REM NUMERO DE LINEA, SEGUIDO POR
30 REM UNA PALABRA DE INSTRUCCION.
```

En casi cualquier punto de cada línea de programa puede dejarse cualquier número de espacios en blanco; el aprovechamiento acertado de esta posibilidad mejorará la legibilidad del programa.

2.10 Datos

Una de las virtudes del Basic es la sencillez con que se representan los datos. En un programa escrito en Basic, éstos pueden ser de dos **tipos:** números, o datos **numéricos,** y **cadenas de caracteres,** o datos **alfanuméricos.** La combinación de unos y otros da lugar a diversas **estructuras de datos.**

2.11 Constantes

Una **constante** es un dato con un valor específico, y mantiene ese valor invariable a todo lo largo de la ejecución del programa.

Las **constantes numéricas** pueden tener coma decimal e ir precedi-

das de un signo + o −. He aquí algunas constantes numéricas correctas:

1 −2 3,14159 −57,6213

Las constantes numéricas pueden también escribirse según la notación llamada de **coma flotante:** como producto de un entero o una fracción por una potencia entera de 10. Esta se denota mediante la letra E colocada delante de la potencia de 10. Por ejemplo:

1,2E5 $(= 1,2 \times 10^5 = 120.000)$
3E−4 $(= 3 \times 10^{-4} = 0,0003)$
−5,1E7 $(= 5,1 \times 10^7 = 51.000.000)$

A cada implementación del Basic se asocia un límite en el tamaño de los datos numéricos. La introducción de un valor superior a dicho límite genera un error.

Las **constantes alfanuméricas** son cadenas de caracteres encerrados entre comillas. Ejemplo:

```
"SER, O NO SER?"
"01-969 4219"
""
```

La última cadena no tiene caracteres y se llama **cadena nula;** resulta sorprendentemente útil. Por lo general, el número de caracteres de una cadena cualquiera tiene un límite, que depende de la versión del lenguaje.

2.12 Variables

Los datos se identifican por medio de **variables.** Una variable puede imaginarse, en un sentido matemático, como una letra que equivale a un número, y también como una dirección de la memoria del ordenador en la que se haya contenido el valor de un dato.

En Basic, una variable se representa mediante una sola letra a la que puede seguir una sola cifra. Si la variable es alfanumérica, a dicha cifra sigue el símbolo $. He aquí algunos ejemplos de variables Basic correctas:

Numéricas:	`A`	`K3`	`P0`
Alfanuméricas:	`B$`	`J5$`	`T0$`

e incorrectas:

Numéricas:	`A10`	`HT`	`3K`
Alfanuméricas:	`A$3`	`$A3`	`AB$`

2.13

Comentarios

En Basic, pueden insertarse observaciones en cualquier punto del programa. Su finalidad es facilitar las cosas al autor del programa o a quienquiera que pueda leerlo. El ordenador no tiene en cuenta las observaciones.

La palabra de instrucción que inicia un comentario es **REM.** Por ejemplo:

```
10 REM ES LA INSTRUCCION PARA UN COMENTARIO
```

2.14

Características dependientes de la implementación

Incluso algunas de las características esenciales presentadas en los párrafos anteriores deben ser modificadas en algunas versiones del Basic. Así, en algunas los nombres de variables alfanuméricas empiezan con **$** —por ejemplo, **$A**— y en otros el nombre de la variable puede tener más de una letra. La única característica de algunas versiones que se usará a veces en este libro es la reunión de varias instrucciones en una sola línea, ya que se trata de una posibilidad común a la mayoría de las versiones para microordenador del Basic.

Las diferentes instrucciones que forman la línea se separan mediante el signo dos puntos. Por ejemplo:

```
50 LET T = 0 : REM INICIALIZA EL TOTAL DE
                 VENTAS
```

Observe que sólo hay un número de línea.

En muchas versiones puede prescindirse de la palabra **LET** y sustituir **PRINT** por un signo de interrogación, aunque en este libro no seguiremos ninguna de tales opciones.

2.15

Conclusión

En este capítulo hemos estudiado unos pocos conceptos muy importantes, en particular los de lenguajes de referencia y de implementación. También hemos discutido algunos aspectos esenciales del

Basic, en concreto las estructuras de programa y línea y la representación de datos.

Los puntos más importantes pueden resumirse como sigue:

- Para resolver el problema de las diferentes versiones del Basic creadas para los diversos ordenadores, en este libro se usará el lenguaje de **referencia,** que contiene todos los rasgos esenciales del Basic y a la vez es independiente de cualquier ordenador. Las versiones creadas para los diversos ordenadores se llaman **implementaciones.**
- Un programa en Basic está formado por una serie de **sentencias,** cada una de las cuales empieza por un **número de línea** al que sigue una **palabra de instrucción.**
- Los programas en Basic admiten dos tipos de datos: **numéricos** y **alfanuméricos.**
- Un dato provisto de un nombre se llama **variable.** Cuando un dato toma un valor concreto fijo, se llama **constante.**
- Para facilitar la legibilidad del programa, puede insertarse en cualquier punto del mismo una observación, que se identifica mediante la palabra **REM.** El ordenador no tiene en cuenta las observaciones.

2

Ejercicio

1. Defina brevemente los siguientes términos: lenguaje de referencia; implementación; sentencia; datos numéricos y alfanuméricos; constante; variable; cadena nula; interactivo.
2. Identifique, dentro de la siguiente lista de nombres de variables en Basic, los que son incorrectos:

```
K3, A14, MO, A$2, 3A, $M, AB, J5$, Q
```

3. Explique qué diferencias hay entre las constantes 6 y "6".
4. Indique algunas de las aplicaciones de las observaciones dentro de un programa.
5. Resuma el desarrollo del lenguaje Basic.
6. Haga una lista de todas las diferencias que ha detectado al estudiar este capítulo entre el lenguaje de referencia y la implementación que utiliza su ordenador.
7. Pase a forma decimal los siguientes números escritos en notación de coma flotante:

```
4E5, .61E8, -.5312E10, .7E-3, -.916E-5
```

8. Pase los siguientes números decimales a notación de coma flotante:

```
637000, .002291, 4.631, -0.00584, -6387900
```


RESET
RETURN

3 Entrada, proceso, salida

Todos los programas de ordenadores tienen tres partes: **entrada, proceso** y **salida.** La entrada es el conjunto de datos que se proporcionan al ordenador; el proceso consiste en la manipulación o tratamiento de esos datos; y la salida es la visualización o impresión de los resultados del proceso. En algunos programas, la entrada, el proceso y la salida se producen en fases claramente diferenciadas. En otros, sobre todo en los llamados interactivos, las fases de entrada, proceso y salida se repiten con frecuencia a lo largo de la realización del programa.

En este capítulo presentaremos las instrucciones del lenguaje Basic que gobiernan las operaciones mencionadas. Primero veremos la entrada y la salida, dejando el proceso para el final.

Algunos programas tienen dos facetas suplementarias llamadas **almacenamiento** y **recuperación** de datos. El almacenamiento suele efectuarse en cintas o discos magnéticos; trataremos estos aspectos de la programación en el capítulo 10.

3.1 Entrada

Se llama entrada al aporte de datos a un ordenador desde el entorno que rodea al mismo. Habitualmente las fuentes de datos son

diversas y la operación de entrada de los mismos es realizada mediante diferentes dispositivos que, en el caso de trabajar con el lenguaje Basic, se reducen a tres: el teclado del ordenador o de un terminal, el programa y los archivos creados en cintas o discos magnéticos. Los primeros dos procedimientos se analizarán en este mismo capítulo; el estudio de los archivos queda para el capítulo 10.

3.2 Entrada por teclado

Es el procedimiento más común de introducción de datos, y por lo general se realiza mientras el programa está en marcha. Esta entrada es de naturaleza **interactiva**, porque permite la "conversación" entre la persona sentada ante el teclado y el programa.

La instrucción que posibilita la entrada por teclado es **INPUT,** a la que siguen los nombres de las variables cuyos valores han de ser cargados. Por ejemplo: si se utiliza la instrucción

```
10 INPUT N$,R
```

y se introducen mediante el teclado los datos

```
A.M. RODRIGUEZ, 2.79
```

la variable alfabética **N$** adoptará el valor **A.M. RODRIGUEZ** y la variable numérica **R** se convertirá en **2,79.**

Cuando se responde a una instrucción **INPUT** no suele ser necesario encerrar los datos alfabéticos entre comillas, pero el programa advertirá un error si esos mismos datos se introducen como correspondientes a una variable numérica.

3.3 Entrada de datos cargados con el programa

En algunos casos, los datos deben cargarse en el ordenador junto con el propio programa que ha de procesarlos, quizá porque aquél no soporte terminales interactivos, o porque los datos representan únicamente elementos de referencia que no cambien de un pase del programa a otro. En estos casos, los datos se introducen por medio de una instrucción **DATA** y se introducen con **READ.** Así, las sentencias

```
10 READ A, B, C
50 DATA 3, -2, 8.6
```

hacen que la variable **A** tome el valor 3, la **B** −2 y la **C** 8,6.

Teniendo en cuenta que a cada variable corresponde un dato, pueden usarse varias instrucciones **READ** con una sola **DATA,** y viceversa. Así, las series de sentencias

```
10 READ A
20 READ B,C
50 DATA 3, -2
60 DATA 8.6
```

dan el mismo resultado que la serie anterior.

Una misma variable puede adoptar diferentes valores en diferentes partes de un programa. Por ejemplo: las instrucciones

```
 10 READ P$
 30 READ P$
 50 READ P$
100 DATA "OOX", "XOX", "XXX"
```

dan a la variable **P$** los valores **OOX** en la línea 10, **XOX** en la 30 y **XXX** en la 50. Observe las comillas de la sentencia **DATA:** son características de la mayoría de las versiones del Basic.

Las sentencias **DATA** pueden aparecer en cualquier punto de un programa, aunque es práctica habitual situarlas a continuación de la sentencia **READ** a que hacen referencia o al final del programa.

3.4 La instrucción RESTORE

La instrucción **RESTORE** permite al programa "volver a empezar" a trabajar desde el principio de la lista de valores contenidos en la sentencia **DATA** del mismo. Así, las sentencias

```
 10 READ P , Q
 20 RESTORE
 25 READ S, T
100 DATA 7.5, 3E5
```

asignan a las variables **P** y **Q** los valores 7,5 y 3E5 ($=3 \times 10^5 = 300.000$); tras la instrucción **RESTORE,** las variables **S** y **T** toman los mismos valores. Si lo piensa un poco, comprenderá fácilmente que no tiene sentido utilizar una instrucción **RESTORE** en combinación con otra **INPUT.**

3.5 Salida

Se llama salida a la operación de entrega de datos por un ordenador al entorno inmediato del mismo. Se realiza mediante

diversos dispositivos, pero, en el caso de los programas escritos en lenguaje Basic, las posibilidades se limitan a la impresora, la pantalla y los archivos de datos. La salida hacia un archivo de datos se tratará en el capítulo 10. Las otras dos modalidades se discutirán en la próxima sección.

3.6

Salida a una impresora o una pantalla

Para imprimir o visualizar en pantalla los datos de salida se emplea la instrucción **PRINT**. Que la salida se produzca a través de la pantalla, de la impresora o de ambas depende de la configuración física del ordenador. Por lo general, el programa Basic no controla este extremo.

La palabra **PRINT** puede ir seguida por una lista de nombres de variables, cuyos valores constituirán la salida. Por ejemplo: si **X** = 3,7, **Y$** = "**O**" y **Z** = 3,8, la sentencia

```
20 PRINT X, Y$, Z
```

producirá la salida

```
3 .7            O               3.8
```

Cuando en una sentencia **PRINT** se usan comas entre los nombres de las variables, la salida aparece distribuida en columnas. El número y la anchura de dichas columnas depende del ordenador que se esté usando.

Si se usa punto y coma en lugar de coma entre los nombres de las variables, los datos de la salida aparecen un poco más juntos. Por ejemplo: con los mismos valores y variables de arriba, la sentencia

```
20 PRINT X; Y$; Z
```

da lugar a la salida

```
3 .7 O 3.8
```

Cada sentencia **PRINT** inicia la presentación de la salida en una línea nueva. Sin embargo, si termina en coma o punto y coma, la salida de la siguiente sentencia **PRINT** continuará en la misma línea, sea en columnas, sea en espaciado reducido. Así, si adjudicamos los valores 1 a 8 a las variables, la serie de sentencias

```
10 PRINT A, B,
20 PRINT C, D
30 PRINT E, F,
40 PRINT G, H,
```

producirán la salida

```
1              2              3              4
5              6              7              8
```

Las sentencias **PRINT** admiten el uso de constantes, muy útiles en la confección de encabezamientos, unidades e instrucciones. Por ejemplo: si la variable **T** toma el valor 3,9, la sentencia

```
30 PRINT "TIEMPO TRANSCURRIDO:";T;" MINUTOS
```

da lugar a la salida

```
TIEMPO TRANSCURRIDO: 3.9  MINUTOS
```

También pueden usarse constantes numéricas en las sentencias **PRINT**. Así, la sentencia

```
50 PRINT 1, 2, 3
```

produce el resultado

```
1              2              3
```

3.7
Programa ejemplo 3.1

Este sencillo programa ilustra el empleo de las sentencias **INPUT**, **READ**, **DATA** y **PRINT**.

```
10 REM PROGRAMA EJEMPLO 3.1
20 PRINT "CUAL ES TU NOMBRE?"
30 INPUT N$
40 READ D$
50 PRINT
60 PRINT "HOLA ";N$;". ";
70 PRINT "LA FECHA DE HOY ES ";D$
80 DATA " 17 DE JULIO DE 1980"
90 END
```

La ejecución de este programa presentaría en pantalla el siguiente resultado:

```
CUAL ES TU NOMBRE?
CARLOS

HOLA CARLOS. LA FECHA DE HOY ES 17 DE JULIO DE 1980
```

Se ha subrayado el dato introducido por teclado.

Puntos de interés

- El nombre, variable **N$**, se ha introducido por medio del teclado, mientras que la fecha, variable **D$**, se ha leído en la sentencia **DATA**.
- El resultado de la línea 50 es una línea en blanco.
- La instrucción **END** señala el final del programa.

3.8

Tabulación de la salida

El espaciado de los elementos de la salida puede controlarse en la sentencia **PRINT** mediante la función **TAB**, que especifica el número de caracteres que van desde el inicio de una línea hasta el inicio del siguiente elemento de la salida. Si se usa varias veces en la misma instrucción **PRINT**, el número de columnas se cuenta en todos los casos a partir del inicio de la línea. Es práctica habitual emplear la función **TAB** con punto y coma para separar nombres de variables en la sentencia **PRINT**.

Por ejemplo, si la variable **D$** tiene el valor 0123456789 y las **A$**, **B$** y **C$** los **A**, **B** y **C**, respectivamente, las sentencias

```
20 PRINT TAB(10);"1";TAB(20);"2"
30 PRINT D$;D$;D$
40 PRINT TAB(8);A$;TAB(16);B$;TAB(24);C$
```

producirán el resultado

```
          1         2
012345678901234567890123456789
        A       B       C
```

La función **TAB** es muy útil para alinear encabezamientos con columnas de datos. Por ejemplo:

```
20 PRINT TAB(5); "FECHA";
TAB(15); "IMPORTE"; TAB(25); "BALANCE"
30 PRINT TAB(5); D$; TAB(15); A; TAB(25); B
```

produce el resultado

```
FECHA      IMPORTE   BALANCE
17/07/84   35.00     279.61
```

con los valores **D$**, **A** y **B** con el contenido indicado.

El número de espacios especificados para una función **TAB** puede

ser variable. Así, **TAB(X)** cuenta X espacios a partir del comienzo de la línea; esta posibilidad permite emplear la función **TAB** para realizar gráficos, entre otras cosas.

3.9 Programa ejemplo 3.2

El programa propuesto a continuación crea un formulario que debe rellenarse en la pantalla. En dicho formulario se pide a una persona que indique su nombre, su fecha de nacimiento y su dirección, y la información obtenida se presenta a continuación en pantalla para ser verificada.

Variables

N$	Nombre.
A$	Apellido.
D1$	Dirección, primera línea.
D2$	Dirección, segunda línea.
D3$	Dirección, tercera línea.
P$	Código postal.
D$	Día (de la fecha de nacimiento).
M$	Mes (de la fecha de nacimiento).
AN$	Año (de la fecha de nacimiento).

Programa

```
100 REM PROGRAMA EJEMPLO 3.2
105 REM ENTRADA E IMPRESION DE NOMBRE,
107 REM FECHA DE NACIMIENTO
110 REM Y DIRECCION
115 REM
120 PRINT
125 PRINT "POR FAVOR ESCRIBA SU NOMBRE"
130 PRINT "         EL APELLIDO PRIMERO";
135 INPUT A$
140 PRINT "             AHORA EL NOMBRE";
145 INPUT N$
150 PRINT
155 PRINT "AHORA ESCRIBA SU FECHA DE NACIMIENTO"
160 PRINT"                     PRIMERO EL ANO";
165 INPUT AN$
170 PRINT "                      AHORA EL MES";
175 INPUT M$
180 PRINT "                      AHORA EL DIA";
```

```
185 INPUT D$
190 PRINT
195 PRINT "AHORA ESCRIBA SU DIRECCION"
200 PRINT "                         PRIMERA LINEA";
205 INPUT D1$
210 PRINT "                         SEGUNDA LINEA";
215 INPUT D2$
220 PRINT "                         TERCERA LINEA";
225 INPUT D3$
230 PRINT "                         CODIGO POSTAL";
235 INPUT P$
240 PRINT
245 PRINT "GRACIAS.  AQUI TIENE LA INFORMACION"
250 PRINT "QUE NOS HA FACILITADO, PARA SU COMPRO
           BACION:"
255 PRINT
260 PRINT "NOMBRE"; TAB(25); N$; " "; A$
265 PRINT
270 PRINT "FECHA DE NACIMIENTO"; TAB (25); D$; "
            DE "; M$; " DE "; AN$
275 PRINT
280 PRINT "DIRECCION"; TAB  (25); D1$
285 PRINT TAB (25); D2$
290 PRINT TAB (25); D3$; " "; P$
295 PRINT
300 END
```

Ejecución

```
POR FAVOR ESCRIBA SU NOMBRE
        EL APELLIDO PRIMERO? RODRIGUEZ
            AHORA EL NOMBRE? FRANCISCO

AHORA ESCRIBA SU FECHA DE NACIMIENTO
                     PRIMERO EL ANO? 1958
                       AHORA EL MES? JULIO
                       AHORA EL DIA? 8

AHORA ESCRIBA SU DIRECCION
                      PRIMERA LINEA? CALLE DE LA VICTORIA 17
                      SEGUNDA LINEA? NAVALUENGA
                      TERCERA LINEA? AVILA
                      CODIGO POSTAL? 22065


             GRACIAS.  AQUI TIENE LA INFORMACION
             QUE NOS HA FACILITADO, PARA SU COMPROBACION:

             NOMBRE                   FRANCISCO RODRIGUEZ

             FECHA DE NACIMIENTO      8 DE JULIO DE 1958

             DIRECCION                CALLE DE LA VICTORIA 17
                                      NAVALUENGA
                                      AVILA 22065
```

Puntos de interés

- La información proporcionada por el usuario aparece subrayada.
- Observe que el punto y coma con que terminan algunas sentencias **PRINT** hace que las entradas sucesivas queden en la misma línea.
- Observe los espacios en blanco de las líneas 260, 270, 290.
- En la práctica, esta porción de programa estaría seguida por otra destinada a permitir al usuario la corrección de los posibles datos incorrectos.
- Una porción de programa como la ilustrada tiene aplicaciones muy variadas.

3.10 Proceso

La mayor parte del resto de este libro se dedica al estudio de las diversas modalidades de proceso de datos. En el centro de todas las operaciones de proceso hay una instrucción que asigna un valor a una variable, operación conocida como **asignación.** En Basic, la instrucción de asignación es **LET**.

Una sentencia **LET** siempre incluye un signo igual y una única variable a la izquierda del mismo. A la derecha del signo igual hay una constante, una variable o una expresión. Veamos algunos ejemplos correctos de sentencias **LET**:

```
10 LET A = 0
20 LET X$ = "KILOGRAMOS"
30 LET J = M
40 LET T = W + 3
50 LET K = K + 1
```

La sentencia **LET** funciona como sigue: el valor de la constante, la variable o la expresión situadas a la derecha del signo igual se asigna a la variable que está a la izquierda de dicho signo (por eso sólo puede haber una variable en el lado izquierdo).

Si a los dos lados del signo igual aparece la misma variable, se usa el valor que tenga en ese momento para calcular la expresión de la derecha, y a continuación se asigna el resultado a la variable, que adquiere así un nuevo valor. En el último de los anteriores ejemplos, si **K** vale 3, la expresión **K** + 1 situada a la derecha del signo igual valdrá 4, que pasa a convertirse en el nuevo valor de **K**.

Recogemos a continuación algunos ejemplos frecuentes de sentencias **LET** incorrectas:

`10 LET A = "ELLA"`	(variable numérica con valor alfabético)
`20 LET X + 1 = Y`	(expresión al lado izquierdo del signo igual)
`30 LET J$ = 5.4`	(variable alfabética con valor numérico)

En muchas versiones del Basic puede omitirse la palabra **LET** en las sentencias de asignación. La versión de referencia no sigue esta norma.

3.11

Expresiones aritméticas

En Basic, las expresiones aritméticas son muy parecidas a las usadas en matemáticas. He aquí algunas fórmulas corrientes presentadas en forma matemática y como sentencias **LET** en Basic:

Matemáticas	*Basic*
$I = \frac{PRT}{100}$	`LET I = P*R*T/100`
$a = \frac{1}{2} bh$	`LET A = B*H/2`
$p = 2(l + b)$	`LET P = 2*(L + B)`
$A = P\left(1 + \frac{R}{100}\right)^T$	`LET A = P*(1 + R/100) ^ T`

Observe que algunos símbolos aritméticos adoptan en Basic una forma algo diferente. La tabla indica las equivalencias:

	Matemáticas	*Basic*
Adición	+	+
Sustracción	−	−
Multiplicación	×	*
División o fracción	÷ o ——	/
Elevación a una potencia		↑
Paréntesis	( )	( )

El orden en que se ejecutan las diversas operaciones es igual en Basic que en aritmética:

1. elevación a una potencia;
2. multiplicación o división;
3. adición o sustracción;
4. lo primero que se ejecuta, siempre en el orden que acaba de indicarse, son las expresiones encerradas entre paréntesis.

3.12 Programa ejemplo 3.3

Este programa es un sistema calculador de interés compuesto. Solicita la entrada de una cantidad ahorrada, de un interés y de un tiempo de imposición y produce como salida la cantidad acumulada tras dicho tiempo de imposición.

Variables

P Cantidad ahorrada.
I Interés.
T Tiempo de imposición.
A Cantidad acumulada.

Programa

```
100 REM PROGRAMA EJEMPLO 3.3
105 REM CALCULOS DE INTERES COMPUESTO
110 REM
115 PRINT "CALCULOS DE INTERES COMPUESTO"
120 PRINT
125 PRINT
130 PRINT "CANTIDAD AHORRADA (PESETAS)";
135 INPUT P
140 PRINT "INTERES";
145 INPUT I
150 PRINT "TIEMPO DE IMPOSICION";
155 INPUT T
160 PRINT
165 LET A = P*(1 + I/100)^T
170 PRINT "DINERO ACUMULADO :";A;" PESETAS"
175 PRINT
180 END
```

Ejecución

```
CALCULOS DE INTERES COMPUESTO


CANTIDAD AHORRADA (PESETAS)? 3000
INTERES? 11
TIEMPO DE IMPOSICION? 5

DINERO ACUMULADO : 5055.17  PESETAS
```

Puntos de interés

Observe la inclusión de las unidades en las líneas 130 y 170. En el primer caso solicitan del usuario la introducción de una cantidad de dinero y en el segundo indican la cantidad que aparecerá como salida.

3.13 Secuencia de varias sentencias en una sola línea

Uno de los principios implícitos en la versión original del Basic fue el de una sola operación por línea de programa, y la versión de referencia se atiende a dicho principio.

Sin embargo, esa restricción da lugar en ocasiones a programas largos y engorrosos que el ordenador tarda en traducir y en ejecutar. Por ello hay ahora muchas versiones de Basic que permiten acumular varias sentencias en una sola línea. Pese a que no se trata de una característica original, es demasiado potente como para ignorarla por completo en un análisis del Basic al nivel de profundización de este libro.

Las sentencias sucesivas en una misma línea se separan mediante dos puntos:

```
100 INPUT F: LET I =12*F : PRINT I; "PULGADAS"
```

Observe que, pese a haber un solo número de línea, cada nueva sentencia comienza por una palabra de instrucción.

Pero el número de sentencias que caben en una línea tiene un límite y es preciso decidir cuáles se colocan en una misma línea. Aunque no hay una norma rigurosa, es útil el siguiente criterio:

Coloque varias sentencias en una misma línea si encajan, si hay entre ellas una estrecha conexión lógica y si no se produce entre unas y otras ninguna ruptura del flujo de control.

El último punto se discutirá con más detalle en el próximo capítulo.

3.14 PEEK y POKE

Aunque, con independencia de la estructura del ordenador que lo soporte, el Basic es un lenguaje de alto nivel, hay dos posibilidades de bajo nivel comunes a varias versiones que, en ciertos casos, resultan muy útiles. Estas posibilidades permiten examinar una posición de memoria determinada o almacenar en la misma un dato particular.

La función **PEEK** devuelve el valor decimal del contenido de una posición determinada de la memoria principal del ordenador. Por ejemplo:

```
LET A = PEEK (X)
```

devuelve el valor decimal del elemento de la dirección (decimal) **X**; aunque el valor de **A** es —en la mayor parte de los casos— un entero comprendido entre 0 y 255, hay que recordar que no siempre representa un número, y puede ser también el valor numérico de un código de carácter ASCII, una instrucción de máquina o parte de un número que abarca varias posiciones de memoria.

La instrucción **POKE** sirve para cargar en una posición determinada de memoria un dato específico. Por ejemplo:

```
POKE X,A
```

inserta el valor de **A** en la dirección (decimal) **X**. La variable **A** debe ser un entero comprendido —en la mayor parte de los casos— entre 0 y 255, aunque puede representar el valor numérico de un carácter ASCII o una instrucción de máquina.

En muchos ordenadores, la pantalla está controlada directamente por una parte de la memoria principal, de forma que cualquier carácter adscrito a una dirección de ese área aparece en la correspondiente posición de la pantalla. Precisamente una de las principales aplicaciones de las funciones **PEEK** y **POKE** es controlar el contenido de la pantalla a través de la memoria.

Las posibilidades **PEEK** y **POKE** no se incluyen en la versión de referencia del lenguaje por dos razones: primero, porque no son elementos característicos de un lenguaje de alto nivel; y segundo, porque dependen del ordenador utilizado. No están presentes en todas las versiones del Basic y su forma precisa difiere de un ordenador a otro. En consecuencia, ninguno de los programas-ejemplo de

este libro utiliza las funciones **PEEK** y **POKE**, aunque quien lo desee puede recurrir a ellas para contestar a las preguntas planteadas en los ejercicios.

3.15
Conclusión

Hemos visto en este capítulo las características elementales de la entrada, el proceso y la salida de datos en Basic. Los puntos más importantes son:

- La entrada por teclado se realiza mediante una sentencia **INPUT**.
- La sentencia **READ** sirve para introducir los datos cargados por otra sentencia **DATA**.
- La salida de datos se obtiene mediante una sentencia **PRINT**.
- La sentencia **LET** asigna un valor a una variable.
- En Basic, las expresiones aritméticas se parecen a las usuales en matemáticas y el orden de ejecución de las operaciones es el mismo en ambos casos.

3
Ejercicio

1. Defina brevemente los términos entrada, proceso, salida y asignación.
2. Examine la siguiente secuencia de sentencias:

```
10 READ A, B
20 READ C, D
30 RESTORE
40 READ C, D
50 DATA 7, 6, 5, 4
```

   a) ¿Cuál es el valor de la variable **C** en la línea 20?
   b) ¿Cuál es el valor de la variable **D** en la línea 40?
3. Identifique los errores cometidos en las siguientes sentencias:

```
10 PRINT "QUE HORA ES?"
20 READ H$;
30 PRINT "SON LAS," H$
```

4. Si **A** = 3 y **B** = 4, ¿qué valores adoptarán tras la sentencia

```
20 LET A =B
```

Si **A** = 3 y **B** = 4, ¿qué valores adoptarán tras la sentencia

```
20 LET B = A
```

Explique sus respuestas.

5. *a*) Escriba una serie de sentencias que intercambien los valores de dos variables.

b) Tres variables, llamadas **A**, **B** y **C**, deben intercambiar sus valores con arreglo a la siguiente regla:

el valor de **A** pasa a **B**,
el valor de **B** pasa a **C**,
el valor de **C** pasa a **A**.

Escriba una serie de sentencias que produzcan el cambio buscado.

6. Indique el valor de la variable **C** tras la realización de la siguiente secuencia de sentencias:

```
100 READ A, B
110 DATA 8E6, 4E3
120 LET C = A/B
```

7. Indique el orden en que se calculan las expresiones propuestas a continuación. La primera no es más que un ejemplo (utilice los valores **A** = 10, **B** = 5, **C** = 8).

a) `3*A/(C - 3) = 3*10/(8 - 3)` primero los paréntesis
`= 3*10/5` luego la multiplicación
`= 30/5` después la división
`= 6`

b) `(A - C) * (B + 9)`
c) `2*A + 3*B`
d) `2*(A + 3)*B`
e) `3*(A - B)^2`
f) `C/(A - 2*B)`

¿Qué cree usted que pasaría si en un programa aparece la última expresión?

8. Escriba un programa con arreglo a las siguientes especificaciones:

El programa sirve para recoger datos destinados a servir a un sistema escolar de registro de alumnos. Tras visualizar las instrucciones adecuadas, se pide la introducción de la información siguiente: nombre (nombre de pila y apellido), clase y cuatro cuestiones opcionales previamente seleccionadas. A continuación han de visualizarse los datos para su verificación.

9. Escriba un programa que acepte como entrada una cantidad de dinero y un tipo de IVA. A continuación ha de calcular el IVA devengado por esa cantidad y presentar el resultado junto con la suma de la misma cantidad más el impuesto.

Puede utilizarse la siguiente fórmula:

$$I = \frac{CT}{100}$$

I: IVA
C: cantidad de dinero
T: tipo del IVA (%)

$$A = C + I$$

A: cantidad más IVA

10. Vuelva a escribir el programa ejemplo 3.3 acumulando varias sentencias en una sola línea. Reflexione con cuidado sobre la distribución de las sentencias.

11. La figura 3.1 ilustra en sección y en planta un depósito cilíndrico que debe construirse en hormigón. Observe que las paredes y el fondo tienen el mismo grosor.

    Escriba un programa que acepte como entradas el radio interior, el grosor de las paredes y la altura interior del tanque y que calcule la capacidad del mismo y el volumen de hormigón necesario para construirlo.

    La fórmula del volumen de un cilindro es

    $V = \pi r^2 h$ V: volumen
    r: radio
    h: altura
    π: 3,14159

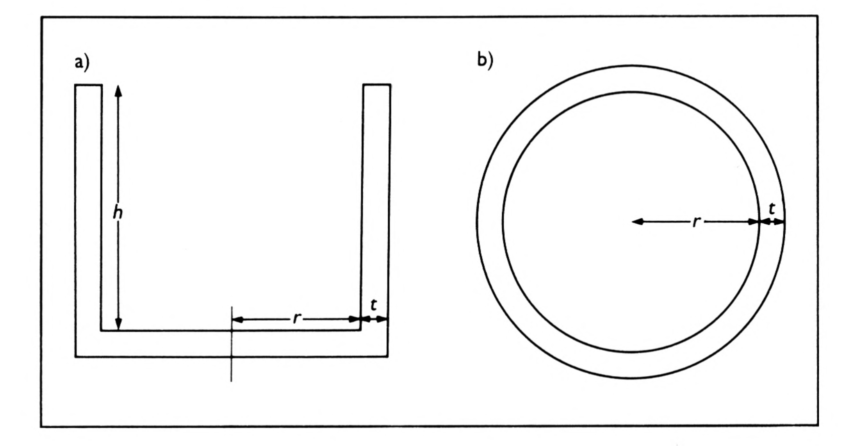


Figura 3.1
El depósito cilíndrico
a) Sección
b) Planta
r: radio interior *t*: grosor
h: altura

12. Construya un programa que acepte cuatro letras cada una en una variable y que dé como salida las 24 palabras que se pueden construir ordenando las letras. Ejecute el programa varias veces hasta determinar qué grupo de cuatro letras es el que da lugar al mayor número de palabras correctas.

13. Utilice como entrada el nombre de una estrella o una galaxia y su distancia al sol en años luz. Transforme la distancia a kilómetros y preséntela como salida en dicha unidad. La velocidad de la luz debe formar parte de los datos cargados en el programa ($2{,}99 \times 10^5$ kilómetros por segundo).

14. Examine la siguiente serie de sentencias:

```
200 LET X= 1024
205 LET A = 64
210 POKE X, A
215 POKE X + 1, A - 1
220 LET B = PEEK (X+1)
225 LET C = PEEK (X)
```

Indique los valores que tomarán **B** y **C** tras haber sido ejecutadas.


RESET
RETURN

4 Bifurcaciones

Uno de los instrumentos más potentes de que disponen los lenguajes de programación es la posibilidad de realizar selectivamente ciertas operaciones cuando se dan ciertas condiciones. Este capítulo estudia esta posibilidad tal como se presenta en el lenguaje Basic.

Dado que la idea de bifurcación condicional es muy compleja, se ha recurrido a un tratamiento paso a paso. La discusión parte de las sencillas bifurcaciones unicondicionales y avanza hacia condicionamientos múltiples, deteniéndose además en algunas construcciones condicionales propias de versiones mejoradas del Basic.

Este es el primer capítulo del libro en que utilizaremos **diagramas de flujo,** que sirven para ilustrar la corriente lógica de control que recorre un programa. Sin embargo, no se consideran como esenciales en el proceso de diseño y redacción de programas. La figura 4.1 recoge los símbolos de flujo que se usarán en el libro.

Lo más importante a tener en cuenta al estudiar el presente capítulo es que resulta vital para entender qué operaciones y en qué condiciones ejecutará un programa. Dentro de las instrucciones del programa, dichas condiciones han de expresarse de la forma más sencilla y clara posible.

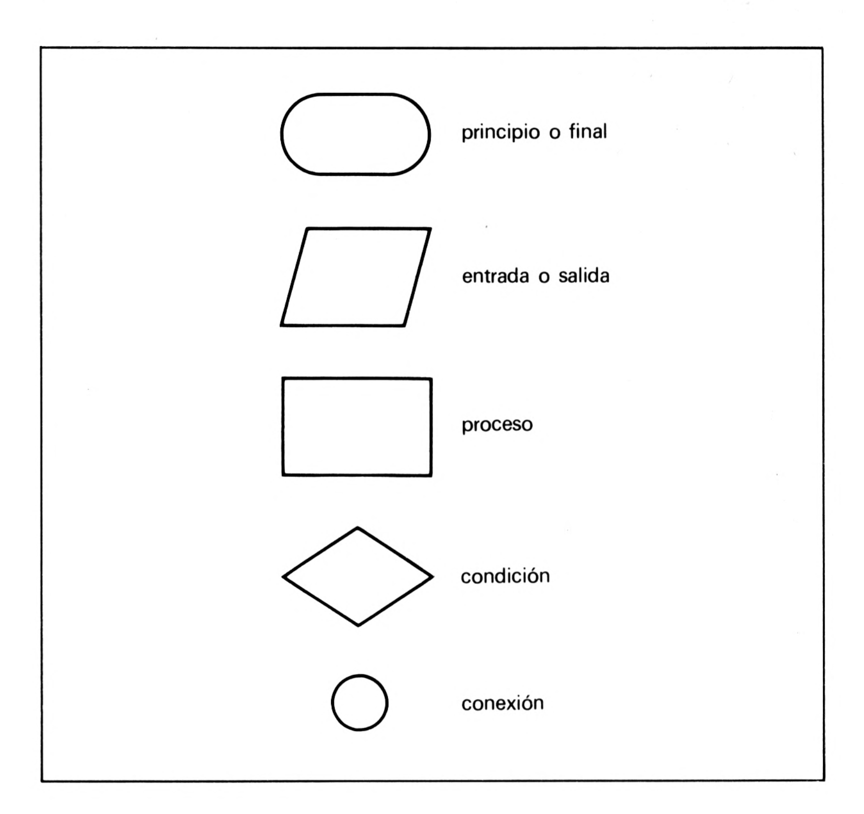


Figura 4.1
Símbolos de flujo

4.1

Operaciones condicionales sencillas

En su sentido más genérico, una operación condicional implica la elección entre dos alternativas más un condicionante que determina la decisión. Por ejemplo:

Si hace buen tiempo, iré a pescar
si no, veré la televisión.

Si la condición (hace buen tiempo) se cumple, la acción realizada es la primera (ir a pescar). Si no se cumple, se desarrolla la segunda actividad (ver la televisión). La figura 4.2 ilustra el proceso en términos de diagrama de flujo.

La forma original del Basic cuenta con dos instrucciones que materializan la idea de bifurcación. Son la instrucción de **bifurcación condicional,** de la forma

IF <condición> **THEN** <número de instrucción>

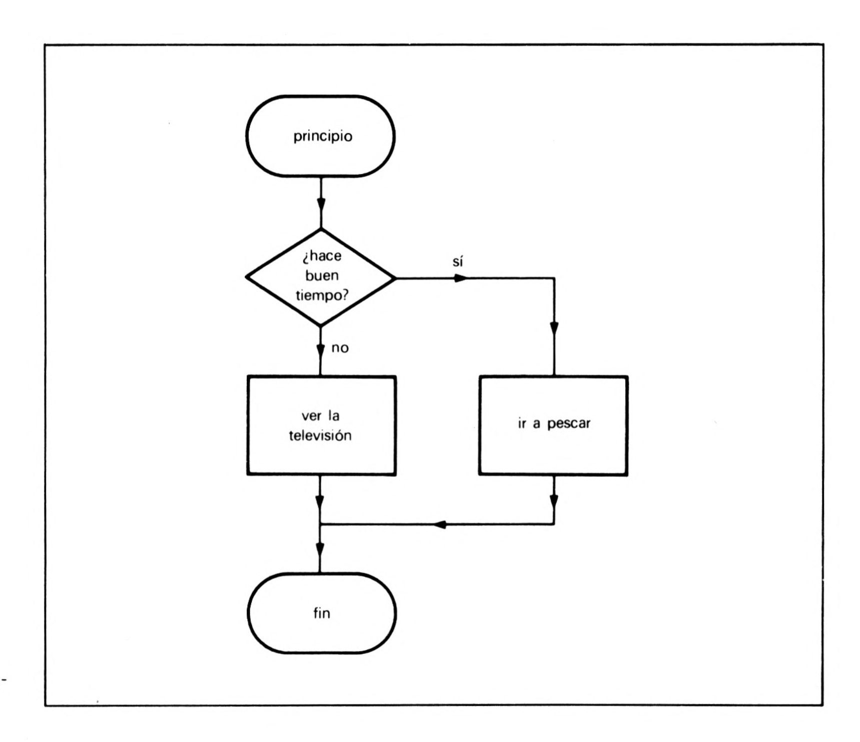


Figura 4.2
Diagrama de flujo de una operación condicional simple

y la de **bifurcación incondicional,** de la forma

GOTO <número de instrucción>.

Con estas instrucciones, el ejemplo de la figura 4.2 se escribiría en Basic como sigue:

```
100 IF W$ = "BUENO" THEN 130
110 PRINT "VER LA TELEVISION"
120 GOTO 140
130 PRINT "IR A PESCAR"
140 END
```

La instrucción **IF** de la línea 100 funciona como sigue:

Si la condición (**W$** = "**BUENO**") se cumple, el control pasa a la línea 130. Si no se cumple, pasa a la siguiente instrucción, en la línea 110.

Aunque esta forma de instrucción **IF** es un poco torpe, es la que corresponde al lenguaje de referencia. En este mismo capítulo veremos versiones más perfeccionadas de la misma.

En una condición pueden usarse diferentes **relaciones:**

=: igual
>: mayor que
<: menor que
>=: mayor o igual que
<=: menor o igual que
<>: distinto a.

4.2

Programa ejemplo 4.1

Muchos programas están pensados para que realicen una serie de tareas diferentes; por lo general, estos programas se controlan mediante **órdenes** introducidas por el usuario. Cada orden traslada el control del programa a la parte encargada de realizar la tarea designada por la orden. Si la orden introducida es irreconocible, aparece en pantalla un mensaje y la solicitud de introducir de nuevo la orden.

La siguiente sección de un programa de control de almacén reconoce las órdenes **RECIBIR**, **ENTREGAR**, **SOLICITAR** y **REALIZADO**, que pasan el control a las líneas 2000, 3000, 4000 y 5000, respectivamente.

Variables

O$ Orden.

Diagrama de flujo

La figura 4.3 ilustra el flujo de control de la sección de programa. Como no se trata de un programa completo, no aparecen los símbolos **PRINCIPIO** y **FIN.**

```
1000 REM PROGRAMA EJEMPLO 4.1
1005 REM SEGMENTO DE UN PROGRAMA DE CONTROL DE
         MERCANCIAS
1010 REM
1015 PRINT "ORDEN?";
1020 INPUT O$
1025 IF O$ = "RECIBIR" THEN 2000
1030 IF O$ = "ENTREGAR" THEN 3000
1035 IF O$ = "SOLICITAR" THEN 4000
1040 IF O$ = "REALIZADO" THEN 5000
```

```
1045 PRINT "ORDEN NO RECONOCIDA. POR FAVOR REPITA
           LA"
1050 GOTO 1020
1055 REM
```

Puntos de interés

- Fíjese en la interconexión de las diferentes condiciones. Si una de ellas no se cumple, el control pasa a la siguiente. Si fallan todas, el control "cae" hasta la instrucción que imprime el mensaje de error.
- Esta combinación de condiciones se llama **cadena de saltos.**

4.3
Condiciones compuestas

En la mayor parte de las versiones del Basic, las condiciones pueden combinarse mediante los operadores lógicos **AND** y **OR** y negarse recurriendo al operador **NOT.** Por ejemplo, la condición para que un número **X** se encuentre dentro del intervalo 1 a 10 (ambos extremos inclusive) es:

```
(X >= 1) AND (X <= 10)
```

Una forma de expresar que dos números **A** y **B** son diferentes sería:

```
NOT (A = B)
```

4.4
Programa ejemplo 4.2

Para ampliar el "vocabulario numérico" de los niños se utiliza un juego de adivinanzas con números. El ordenador "piensa" un número comprendido entre 1 y 100 sirviéndose de su generador de números al azar, y a continuación invita al usuario a adivinarlo.

Para ello proporciona las siguientes pistas:

Diferencia entre el número "adivinado" y el real	*mensaje*
más de 20	**HELADO**
entre 11 y 20	**FRIO**
entre 6 y 10	**TEMPLADO**
entre 2 y 5	**CALIENTE**
1	**TE ESTAS QUEMANDO**

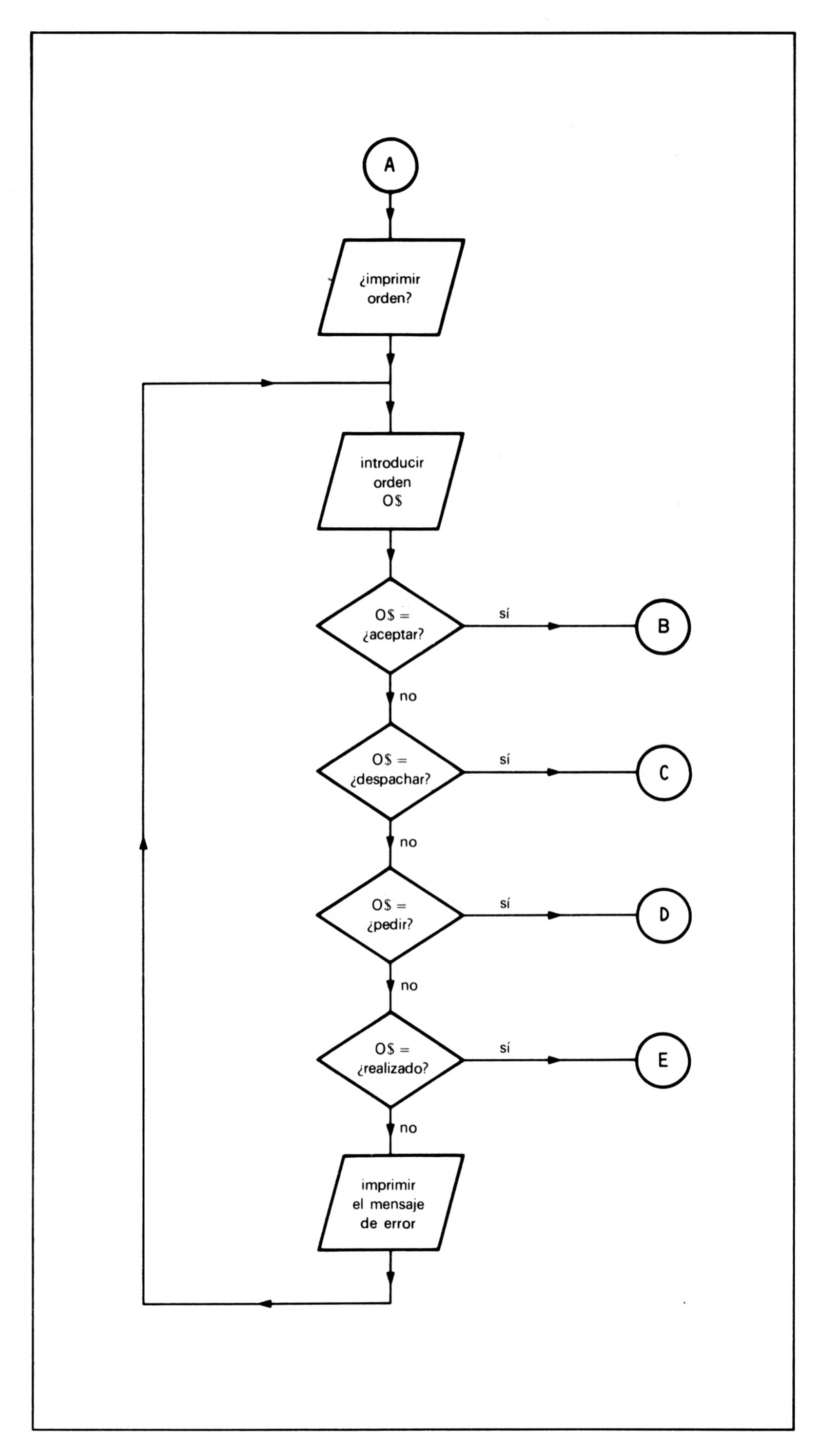


Figura 4.3
Diagrama de flujo del programa ejemplo 4.1

El programa presentado en este ejemplo ilustra un turno del juego, durante el que se van "diciendo" números hasta acertar con el correcto.

Método

Mediante las funciones **RND** e **INT** se genera al azar un entero comprendido entre 1 y 100 (discutiremos estas funciones en el capítulo 8). Se introduce a continuación un número. Si no es un entero comprendido entre 1 y 100, aparece en pantalla un mensaje que solicita la proposición de un nuevo número; esta operación es un ejemplo de verificación o **convalidación** de la entrada. Acto seguido se analiza el número para determinar qué mensaje de los anteriores aparecerá en pantalla. Si el número propuesto es el correcto, el programa termina. En caso contrario, se solicita una nueva propuesta.

Es fundamental que las condiciones tengan un tratamiento sistemático. En este programa se empleó el siguiente método: para determinar si el número introducido es un entero comprendido entre 1 y 100 se emplearon estas condiciones:

```
(G = INT(G)) AND (G >= 1) AND (G <= 100)
```

La función entera **INT** se discutirá en la sección 8.6.

Si el número propuesto es el correcto, aparece en pantalla un mensaje que lo indica y el programa termina. En caso contrario, se calcula la diferencia entre el número propuesto y el real; como no importa el hecho de que la diferencia lo sea por exceso o por defecto, se utiliza la función **ABS** —que se estudiará en la sección 8.3— para que la misma aparezca en forma positiva.

Si esa diferencia es 1, aparece en pantalla el mensaje **TE ESTAS QUEMANDO**; si es menor o igual a 5, el mensaje presentado es **CALIENTE**. El extremo inferior de esta condición queda cubierto por la anterior, y lo mismo es aplicable para el resto de los mensajes.

Variables

N Número real.
P Número propuesto.
D Diferencia entre el número real y el propuesto.

Diagrama de flujo

La figura 4.4 recoge el flujo de control correspondiente a este programa.

Programa

```
100 REM PROGRAMA EJEMPLO 4.2
105 REM JUEGO DE ADIVINANZAS CON NUMEROS
110 REM
200 LET N = INT (100*RND(1)) + 1
205 PRINT "HE PENSADO UN NUMERO"
210 PRINT "COMPRENDIDO ENTRE 1 Y 100. SERIAS"
215 PRINT "CAPAZ DE ADIVINARLO?"
300 INPUT G
305 IF (G = INT(G)) AND (G >= 1) AND (G <= 100)
    THEN 400
310 PRINT "TU RESPUESTA DEBE SER UN NUMERO
          ENTERO"
315 PRINT "COMPRENDIDO ENTRE 1 Y 100."
320 PRINT "POR FAVOR, INTENTALO DE NUEVO"
```

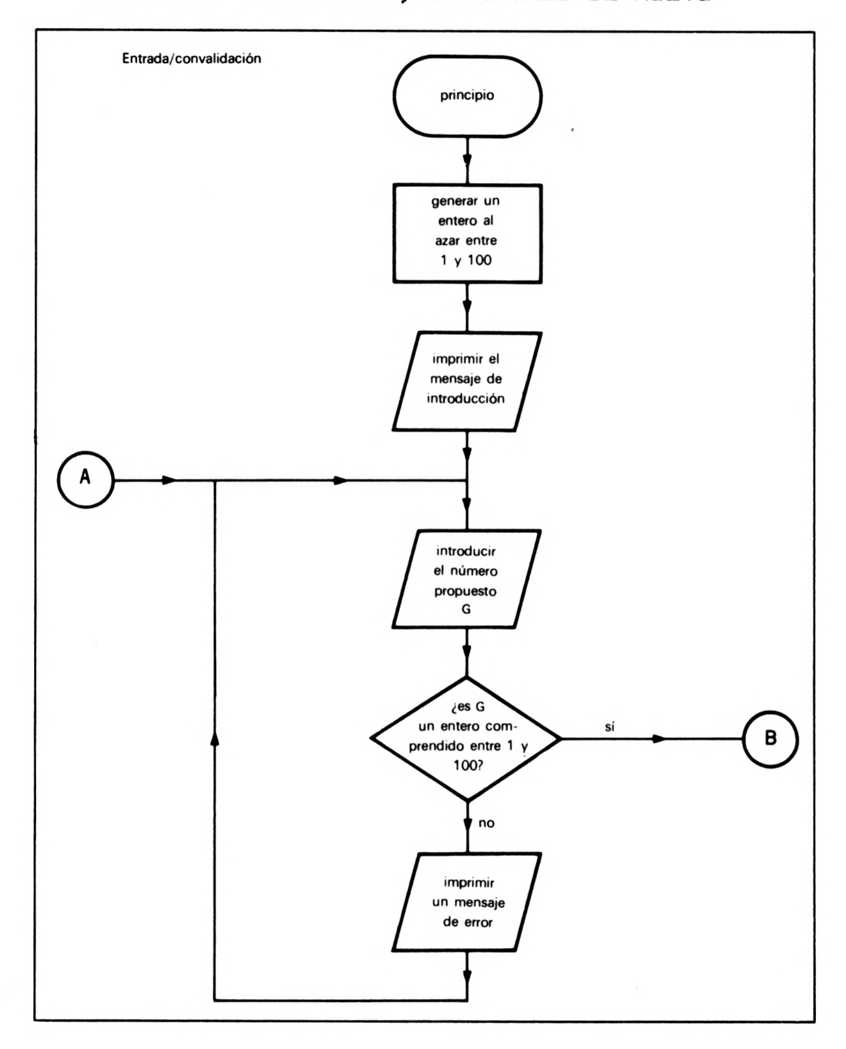
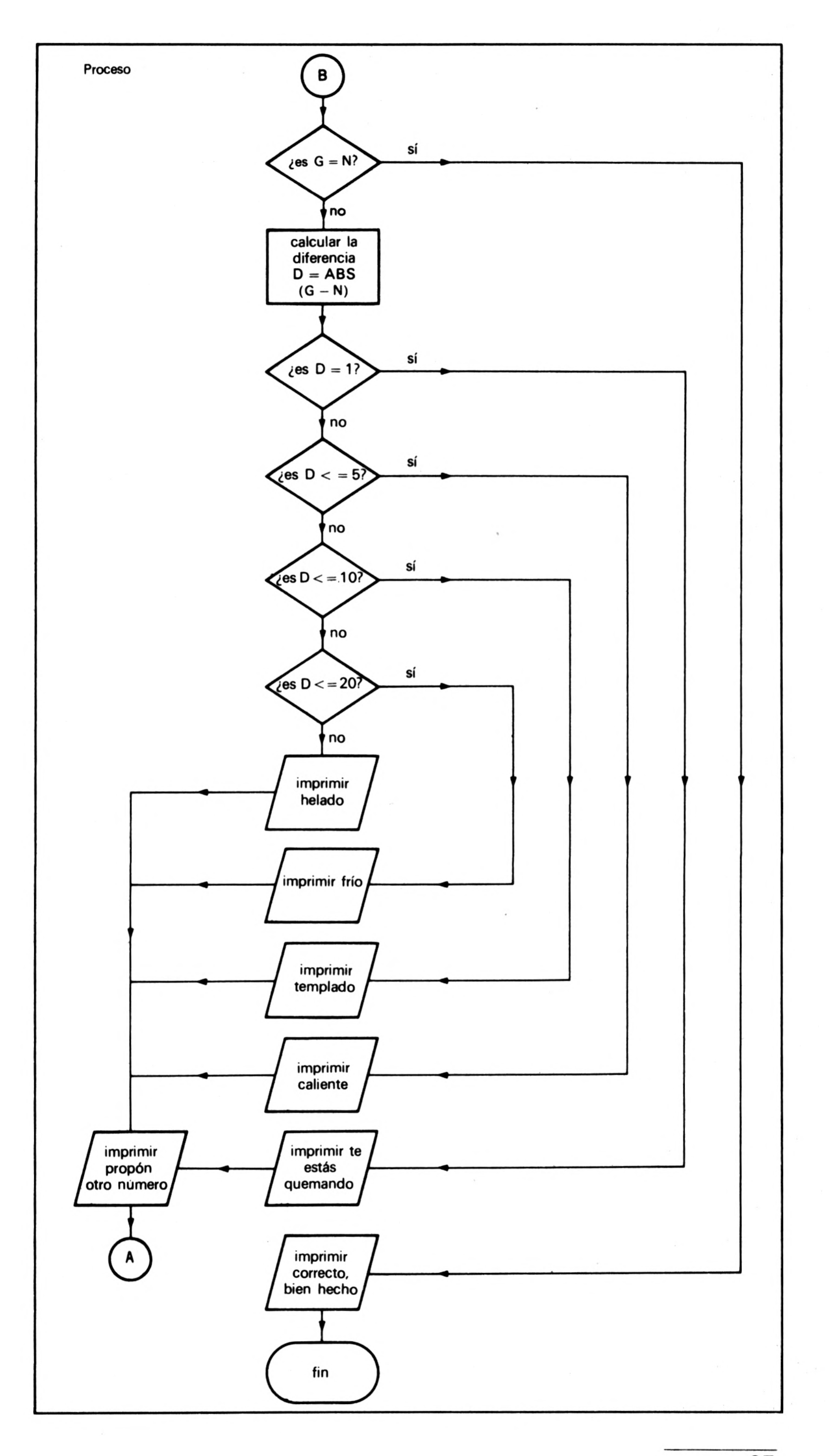


Figura 4.4
Diagrama de flujo del programa ejemplo 4.2



Proceso
B
¿es G = N?
sí
no
calcular la diferencia D = ABS (G – N)
¿es D = 1?
sí
no
¿es D < = 5?
sí
no
¿es D < = 10?
sí
no
¿es D < = 20?
sí
no
imprimir helado
imprimir frío
imprimir templado
imprimir caliente
imprimir te estás quemando
imprimir propón otro número
A
imprimir correcto, bien hecho
fin

```
325 GOTO 300
400 IF G = N THEN 485
405 LET D = ABS(N - G)
410 IF D = 1 THEN 470
415 IF D <= 5 THEN 460
420 IF D <= 10 THEN 450
425 IF D <= 20 THEN 440
430 PRINT "HELADO.";
435 GOTO 475
440 PRINT "FRIO.";
445 GOTO 475
450 PRINT "TEMPLADO.";
455 GOTO 475
460 PRINT "CALIENTE.";
465 GOTO 475
470 PRINT "TE ESTAS QUEMANDO.";
475 PRINT "PROPON OTRO NUMERO."
480 GOTO 300
485 PRINT "CORRECTO. BIEN HECHO"
500 END
```

Puntos de interés

- Observe que el programa salta hasta el primer mensaje (**HELADO**) si no se cumple ninguna de las condiciones.
- Observe que todas las condiciones relativas a los diversos mensajes se han situado juntas.
- La parte común a todos los mensajes (**PROPON OTRO NUMERO**) sólo se escribe una vez, y el control pasa a la instrucción que contiene esa parte desde todas las demás instrucciones de presentación en pantalla.
- Cada uno de los "segmentos" del programa se identifica mediante un número de línea que es múltiplo de 100, lo que deja espacio suficiente para ulteriores modificaciones del mismo.

Ejecución

```
HE PENSADO UN NUMERO
COMPRENDIDO ENTRE 1 Y 100. SERIAS
CAPAZ DE ADIVINARLO?
?30
HELADO.PROPON OTRO NUMERO.
?40
FRIO.PROPON OTRO NUMERO.
?50
TEMPLADO.PROPON OTRO NUMERO.
?55
CALIENTE.PROPON OTRO NUMERO.
?58
TE ESTAS QUEMANDO.PROPON OTRO NUMERO.
?59
CALIENTE.PROPON OTRO NUMERO.
?57
CORRECTO. BIEN HECHO
```

4.5
Bifurcaciones múltiples

Una limitación de la instrucción **IF** radica en que sólo permite elegir entre dos opciones. Pero hay otra instrucción llamada **ON ... GOTO** que permite trabajar con **bifurcaciones múltiples.** Veamos un ejemplo:

```
ON J GOTO 1000, 2000, 3000, 4000
```

En este caso, si **J** tiene el valor 1, el control pasa a la línea 1000, a 2000 si tiene el valor 2, etc. **J** se llama **variable de control,** y en su lugar puede también utilizarse una **expresión de control.**

Los problemas surgen cuando **J** no es un entero positivo comprendido entre 1 y el número máximo de líneas de destino (4 en el ejemplo anterior); cuando esto ocurre, cada ordenador reacciona de una forma diferente, por lo que es aconsejable efectuar algunas verificaciones de la variable de control antes de utilizar la instrucción **ON ... GOTO**.

4.6
Programa ejemplo 4.3

Estudiaremos en esta ocasión un programa de contabilidad creado para poder llevar a cabo cinco transacciones diferentes. El segmento de control del mismo examina el tipo de transacción, y transfiere el control a la parte adecuada del programa. Si el tipo de transacción no corresponde a un entero comprendido entre 1 y 5, aparece en pantalla un mensaje de error.

Método

Se introduce el tipo de transacción y se convalida para determinar si es un entero positivo comprendido entre 1 y 5. Si no lo es, aparece en pantalla un mensaje de error y se introduce otro tipo de transacción. Si el tipo de transacción es válido, se emplea una instrucción de bifurcación múltiple para transferir el control a la parte apropiada del programa.

Variables

T Tipo de transacción.

Diagrama de flujo

La figura 4.5 recoge el diagrama de flujo de este segmento de programa. Observe que no existe un símbolo único que identifique la bifurcación múltiple; lo más aproximado es una cadena de saltos.

Programa

```
100 REM PROGRAMA EJEMPLO 4.3
105 REM SEGMENTO DE UN PROGRAMA DE CONTABILIDAD
110 REM
115 INPUT T
120 IF (T = INT(T)) AND (T >= 1) AND (T <= 5)
    THEN 135
125 PRINT "ENTRADA NO VALIDA. POR FAVOR REPITA"
130 GOTO 115
135 ON T GOTO 500, 1000, 1500, 2000, 2500
140 REM CONTINUA
```

Puntos de interés

- La línea 120 contiene todas las condiciones necesarias para convalidar el tipo de transacción. Si cualquiera de ellas se incumple, el control "cae" hasta el mensaje de error situado en la línea siguiente.
- La línea 135 corresponde a una porción muy extensa del diagrama de flujo del programa.

4.7

Instrucciones condicionales en versiones avanzadas del Basic

Las versiones más avanzadas del lenguaje Basic han ampliado la instrucción **IF ... THEN** a las siguientes formulaciones:

IF <condición> **THEN** <instrucción>

y

IF <condición> **THEN** <instrucción> **ELSE** <instrucción>

Haciendo uso de esta posibilidad, el programa del principio de este capítulo podría redactarse en una sola línea:

```
IF W$ = "BUENO" THEN PRINT "IR A PESCAR"
   ELSE PRINT "VER LA TELEVISION"
```

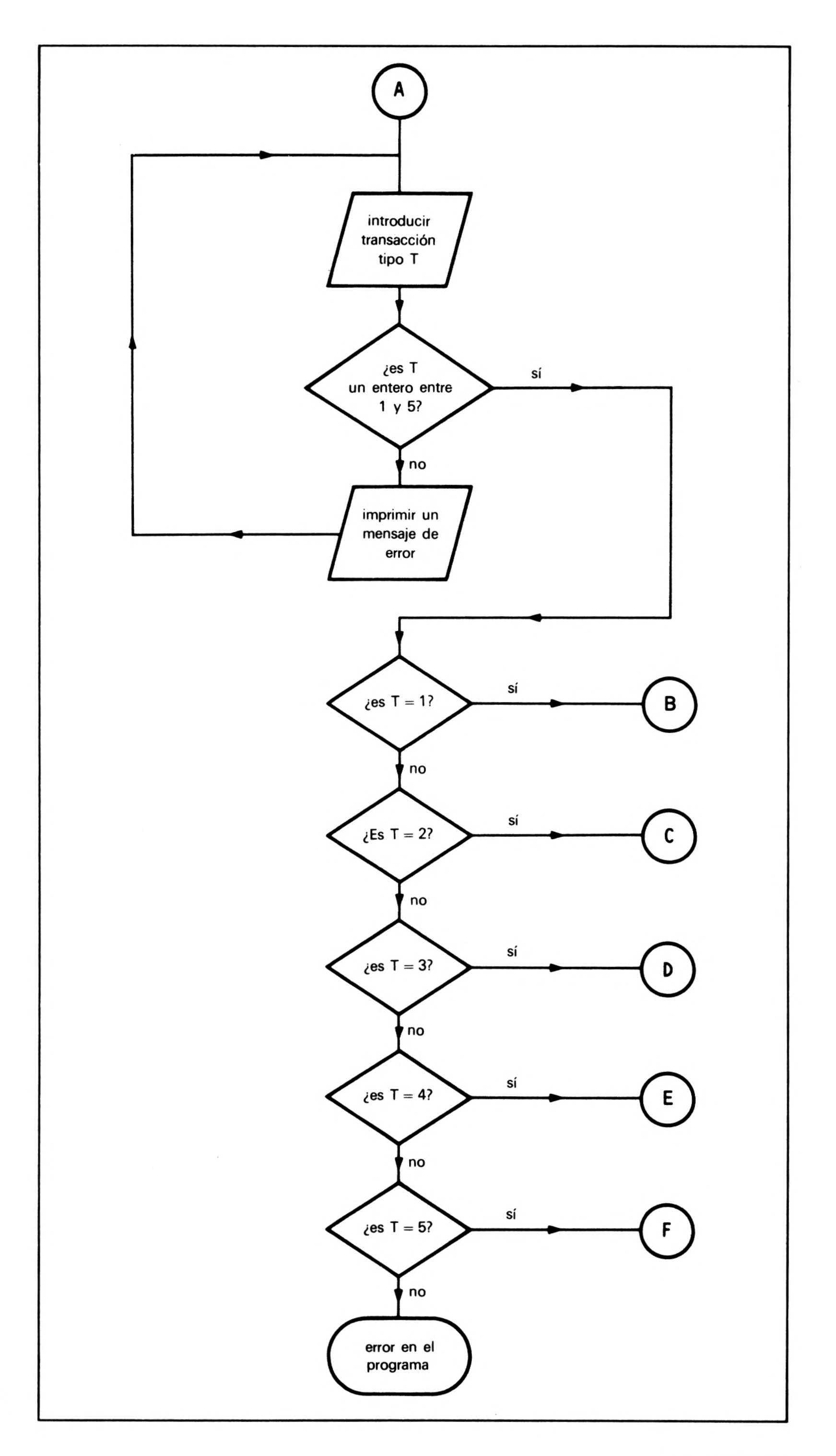


Figura 4.5
Diagrama de flujo del programa ejemplo 4.3

Esta es una opción muy potente que acorta considerablemente el programa, evita la necesidad de escribir varias instrucciones **GOTO** y facilita el seguimiento de la lógica del programa.

Además, tras **THEN** o **ELSE** puede utilizarse una secuencia de instrucciones, lo que da lugar a niveles de eficacia todavía mayores. No obstante, hay que tener en cuenta las limitaciones relativas al número de instrucciones por secuencia mencionadas en el capítulo anterior.

El empleo de condiciones impone otra restricción más a las secuencias de sentencias contenidas en una misma línea: en efecto, si se emplean las instrucciones **IF** o **GOTO**, el control no puede transferirse a las sentencias situadas en el centro de dicha secuencia.

En este libro recurriremos ocasionalmente a estas opciones propias de versiones avanzadas en los programas propuestos a modo de ejemplo. También puede, quien lo desee, resolver los ejercicios mediante las mismas. En cualquier caso, es preciso insistir en que siempre puede redactarse una secuencia de instrucciones sencillas en el lenguaje de referencia que equivalgan lógicamente a una instrucción **IF ... THEN ... ELSE.**

4.8

Programa ejemplo 4.4

Este programa es el mismo de adivinación de números propuesto en el ejemplo 4.2, pero escrito ahora reuniendo varias instrucciones en una sola línea y haciendo uso de la opción **IF ... THEN.**

```
100 REM PROGRAMA EJEMPLO 4.4
105 REM JUEGO DE ADIVINANZAS CON NUMEROS
110 REM ESCRITO EN BASIC AVANZADO
115 REM
120 PRINT "HE PENSADO UN NUMERO"
125 PRINT "COMPRENDIDO ENTRE 1 Y 100. SERIAS"
130 PRINT "CAPAZ DE ADIVINARLO?"
135 LET N = INT(100*RND(1)) + 1
140 INPUT G
145 IF (G = INT(G)) AND (G >= 1) AND (G <= 100)
    THEN 165
150 PRINT "TU RESPUESTA DEBE SER UN NUMERO ENTERO
    "
155 PRINT "COMPRENDIDO ENTRE 1 Y 100. POR FAVOR
          INTENTALO DE NUEVO"
160 GOTO 170
165 IF G = N THEN PRINT "CORRECTO. BIEN HECHO":
    STOP
170 LET D = ABS(N -G)
```

```
175 IF D = 1 THEN PRINT "TE ESTAS QUEMANDO";:
    GOTO 200
180 IF D <= 5 THEN PRINT "CALIENTE";: GOTO 200
185 IF D <= 10 THEN PRINT "TEMPLADO";: GOTO 200
190 IF D <= 20 THEN PRINT "FRIO";: GOTO 200
195 PRINT "HELADO"
200 PRINT "PROPON OTRO NUMERO": GOTO 140
205 END
```

Puntos de interés

- La puntuación de las líneas 175 a 190 es un tanto confusa. El punto y coma forma parte de la instrucción **PRINT**, que hace que la salida permanezca en la misma línea. Los dos puntos separan las instrucciones **PRINT** de las **GOTO**.
- Esta versión del programa es más corta y más fácil de seguir que la anterior.
- En la línea 145 es preferible utilizar la forma convencional de **IF ... THEN.**
- La línea 165 incluye una instrucción nueva, llamada **STOP**. Se utiliza cuando el punto en que se detiene el programa no está en la última línea del mismo.

4.9 Conclusión

En este capítulo hemos presentado los conceptos de bifurcación condicional e incondicional de un programa y su puesta en práctica en Basic. La bifurcación condicional es una característica muy potente, pero debe emplearse con cuidado para que el programa no quede caótico, ilegible o incorrecto.

Los puntos más interesantes de los que se han tocado son:

- La instrucción

  IF <condición> **THEN** <número de línea>

  transfiere el control al número de línea indicado si se cumple la condición. En caso contrario, pasa a la línea situada a continuación de la **IF ... THEN**.
- La instrucción

  GOTO <número de línea>

  transfiere ineludiblemente el control a la línea indicada.

- Pueden establecerse condiciones compuestas mediante los operadores **AND**, **OR** y **NOT.**
- La bifurcación múltiple se crea mediante la instrucción **ON ... GOTO** más una variable o expresión de control.
- Las versiones más elaboradas del Basic permiten la construcción

  IF <condición> **THEN** <instrucción> **ELSE** <instrucción>

- Se llama convalidación a la verificación de los datos de entrada antes de que sean almacenados o procesados.

4

Ejercicio

1. Escriba la siguiente operación condicional en forma de secuencia de instrucciones Basic:

si $a > b$, $x = a$
en caso contrario, $x = b$

2. Exprese las condiciones indicadas a continuación en lenguaje Basic, dando nombres adecuados a las variables cuando ello sea necesario. La primera no es más que un ejemplo.

a) un número situado fuera del intervalo de 1 a 10.

```
(X < 1) OR (X > 10)
```

b) un número situado entre 5 y 20, ambos inclusive.
c) un entero positivo menor que 45.
d) una variable **A$** equivalente a una letra del alfabeto.
Observación: el signo < puede significar "anterior, en orden alfabético, que".
e) un número comprendido entre 48 y 57 o entre 65 y 70.
f) un entero comprendido entre 1 y 5 o mayor que 10.
g) dos números cuya suma sea superior a 100 o inferior a 50.

3. Estudie el segmento de programa en Basic que se presenta a continuación y responda a las preguntas planteadas:

```
500 IF (X >= 5) AND (X <= 10) THEN 540
510 IF (Y < 5) OR (Y > 10) THEN 540
520 LET A = 1
530 GOTO 550
540 LET A = 2
550 REM
```

a) ¿Qué valor adopta **A** si **X** = 3 e **Y** = 2?
b) Si **X** = 6, ¿afectará el valor de **Y** al de **A**?
c) Vuelva a redactar las líneas 500 y 510 como una instrucción única.
d) ¿Qué ocurriría si se intercambiasen las líneas 500 y 510?

e) Vuelva a redactar todo el segmento de programa en una sola instrucción haciendo uso de una versión avanzada del Basic.

4. Escriba un segmento de control de un programa que cumpla las siguientes especificaciones:

Se introduce una orden.
Si la orden es **CARGAR** o **INTRODUCIR**, ir a la línea 10000.
Si la orden es **VISUALIZAR** o **IMPRIMIR**, ir a la línea 5000.
Si la orden es **EJECUTAR** ir a la línea 15000.
En cualquier otro caso, ir a la línea 1500.

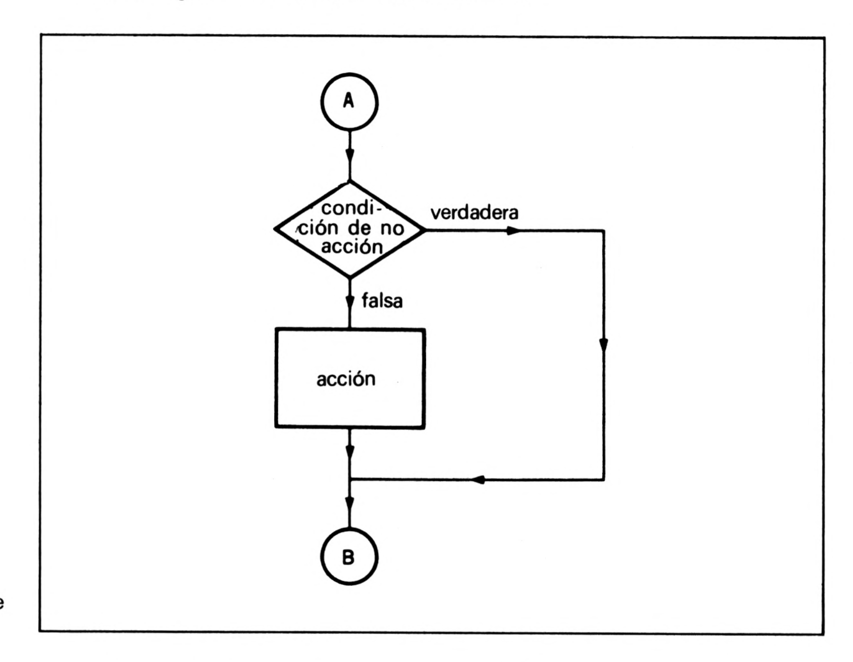


Figura 4.6
Condición para saltarse una parte de un programa

5. Al diseñar un programa, resulta a veces útil crear condiciones que permitan saltar sobre una parte del mismo. La condición utilizada es aquélla bajo la que no se emprende ninguna acción. La figura 4.6 ilustra esta idea.

Escriba porciones de programas que ejecuten las acciones condicionales expresadas a continuación. En todos los casos, dichas condiciones deben "invertirse" para transformarlas en condiciones de no acción. Siga el ejemplo de *a*).

a) Si un número **X** está fuera del intervalo comprendido entre 0 y 100, debe igualarse a 0.
Condición de no acción: que el número esté comprendido entre 0 y 100 (inclusive).

```
50 IF (X >= 0) AND (X <= 100) THEN 60
55 LET X = 0
60 REM CONTINUA
```

b) Si una variable alfanumérica **C$** no es una letra mayúscula, debe adoptar el valor "**?**".

c) Si un número es inferior a 0,000001, debe igualarse a cero.

d) Si la suma de dos números **A** y **B** es mayor que 5.000.000, ambos deben hacerse iguales a 2.500.000.

e) Si dos variables alfabéticas son iguales, ambas deben adoptar el valor "**E**".
f) Si un número es superior a 10, debe hacerse igual a 10, e igual a 1 si es menor que este número.

6. Una ecuación cuadrática o de segundo grado de la forma

$$ax^2 + bx + c = 0$$

tiene dos soluciones:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

y

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

siempre que $b^2 \geqslant 4ac$; en caso contrario no tiene soluciones.
a) Escriba un programa que, al introducir los valores de *a*, *b* y *c* definidos más arriba verifique si la ecuación tiene o no soluciones y, en caso afirmativo, calcule y presente dichas soluciones.
b) El programa escrito en la parte *a*) fallará si $a = 0$ (porque en tal caso la ecuación no es cuadrática). Modifique el programa de manera que presente en pantalla el mensaje **NO CUADRATICA** si la variable *a* es cero.

7. *a*) Modifique el juego de adivinación de números (programa ejemplo 4.2) para que dé únicamente las siguientes pistas:

si el número pensado es mayor que el real **DEMASIADO ALTO**
si el número pensado es menor que el real **DEMASIADO BAJO**

En caso de que el número pensado sea el correcto, ha de aparecer el mismo mensaje del ejemplo.
b) Amplíe el programa original, o la versión reducida, para que pregunte al usuario si desea jugar otra partida y, en caso afirmativo, vuelva a empezar.

8. Escriba un programa que permita utilizar un ordenador como una sencilla calculadora con arreglo a las siguientes normas:

1. Introduzca un número seguido por un signo de operación y por otro número.
2. Si la operación es +, −, * o /, realícela con los dos números introducidos y presente el resultado. Si no es ninguna de las indicadas, presente un mensaje adecuado.
3. Si el usuario desea llevar a cabo otro cálculo, repita el proceso.

9. Por medio de dos variables diferentes, una para las horas y otra para los minutos, introduzca dos horas según la notación de 24 horas. Calcule y presente como resultado la diferencia entre ambas expresada en horas y minutos.

10. Introduzca tres números y presente como resultado el mayor de ellos.


RESET
RETURN

5 Bucles

Estudiaremos en este capítulo uno de los elementos más potentes y más útiles de los lenguajes de programación de alto nivel: **los bucles.** Discutiremos el concepto de bucle de programa y analizaremos varias formas de llevarlo a la práctica en Basic. Se advertirá al lector de los posibles aspectos dependientes del tipo de ordenador que pudieran presentarse.

5.1 La idea de repetición

Un bucle es una porción de programa que se repite. Esto plantea inmediatamente la cuestión de cómo controlar el número de repeticiones que han de efectuarse. La actual técnica de programación dispone para ello de tres recursos:

Repetir *mientras* se cumple cierta condición.
Repetir *hasta que* se cumpla cierta condición.
Repetir *durante* un número de veces especificado.

El Basic sólo dispone de un mecanismo específico del tercer tipo. Los dos primeros recursos se ponen en juego mediante instrucciones

IF ... THEN y GOTO. De todas formas, en este capítulo estudiaremos los tres.

Cuando en un programa se plantea la necesidad de un bucle, la primera decisión que el programador debe tomar se refiere al control del mismo. En ese punto hay que decidirse por uno de los tres mecanismos que acaban de exponerse, aunque en la mayor parte de los casos, la naturaleza del programa hace obvia tal decisión. Pasaremos ahora a la discusión de los medios de control, empezando por el tercero.

5.2 Repetición durante un número de veces especificado

En Basic, la repetición de una parte del programa durante cierto número de veces se controla con las instrucciones **FOR ... TO** y **NEXT**, que incluyen una variable llamada **contador del bucle,** cuyo valor cambia *tras* cada repetición.

Por ejemplo, si el contador es **J**, la instrucción para repetir un bucle 6 veces sería:

```
FOR J = 1 TO 6
```

porción del programa que ha de repetirse

```
NEXT J
```

Observe que el contador aparece tanto en **FOR ... TO** como en **NEXT**.

De una forma más general, para repetir un bucle **N** veces con un contador **X**, se usan las siguientes instrucciones:

```
FOR X = 1 TO N
```

porción del programa que ha de repetirse

```
NEXT N
```

El diagrama de flujo de la figura 5.1 ilustra el proceso.

En algunos ordenadores surgen problemas cuando **N** es igual a cero, es decir, cuando el bucle no debe realizarse ni siquiera una vez. En algunos aparatos, el bucle se ejecuta al menos una vez siempre que está controlado por las instrucciones **FOR ... TO** y **NEXT**. En esos casos, el problema se evita recurriendo a una instrucción **IF ... THEN** para saltar el bucle cuando **N** es cero.

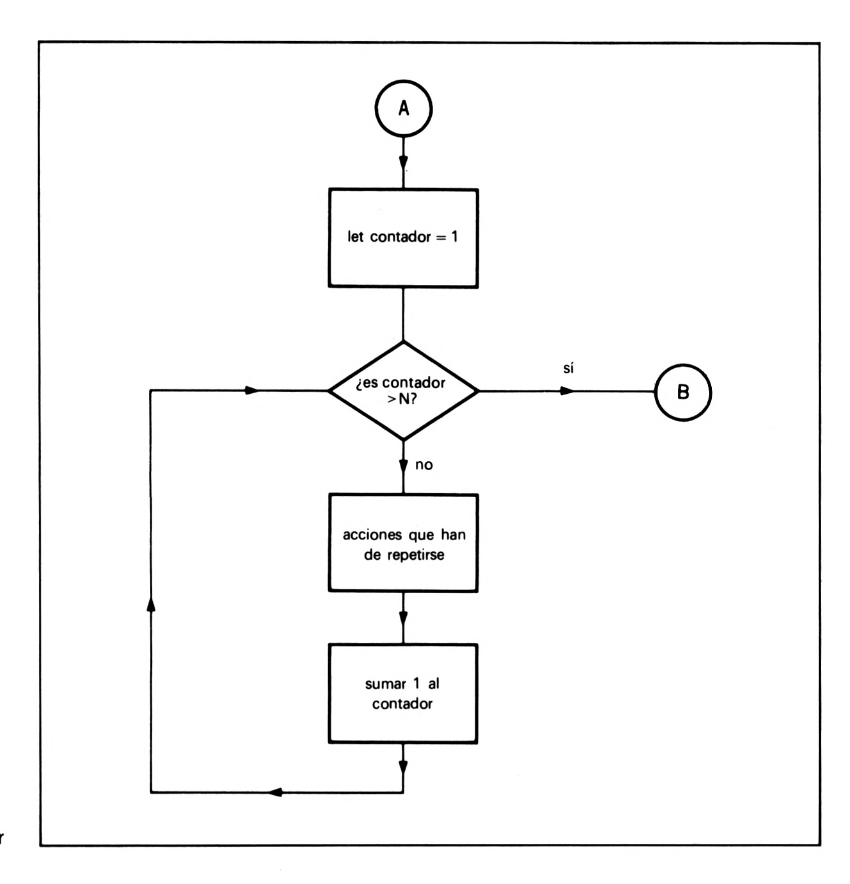


Figura 5.1
Bucle controlado por un contador

5.3
Programa ejemplo 5.1

Este ejemplo ilustra una aplicación sencilla de un bucle controlado por un contador. El programa está pensado para introducir una serie de números y calcular como resultado la media de los mismos.

Método

El programa se basa en el algoritmo siguiente:

Introducir la cantidad de números de la serie.
Poner el total a cero.
Repetir para cada número el siguiente proceso.

Introducir el número.
Sumarlo al total.

Dividir el total por la cantidad de números para obtener la media.
Presentar la media como resultado a la salida.

Variables

N Cantidad de números.
K Contador del bucle.
X Cada uno de los números.
T Total.
M Media.

Diagrama de flujo

Se ilustra en la figura 5.2.

Programa

```
100 REM PROGRAMA EJEMPLO 5.1
105 REM INTRODUCCION Y CALCULO DE LA MEDIA DE
         UNA SERIE DE NUMEROS
110 REM
115 PRINT "DE CUANTOS NUMEROS SE VA A CALCULAR
          LA MEDIA?";
120 INPUT N
125 PRINT "INTRODUZCA LOS NUMEROS"
130 LET T = 0
135 FOR K = 1 TO N
140 INPUT X
145 LET T = T + X
150 NEXT K
155 LET M = T/N
160 PRINT "MEDIA"; M
165 END
```

Puntos de interés

- La variable **X** toma sucesivamente el valor de cada uno de los números introducidos.
- El programa no está protegido frente a valores de **N** negativos o nulos.

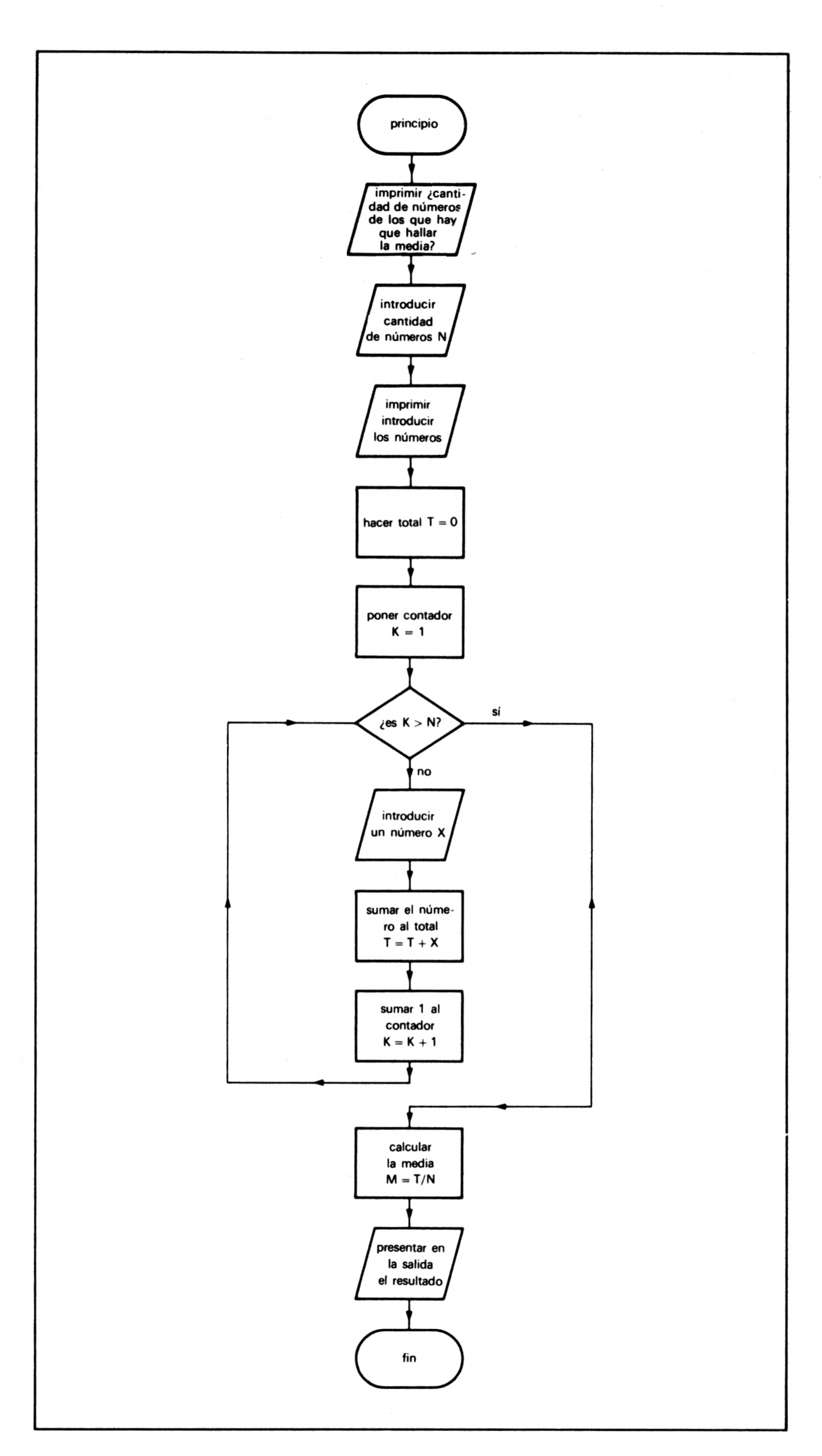


Figura 5.2
Diagrama de flujo del programa ejemplo 5.1

5.4

Forma general de la instrucción FOR ... TO

En los ejemplos anteriores se ha supuesto que el contador incrementa su valor en una unidad tras cada ejecución del bucle, pero en realidad puede incrementarse en cualquier valor, y a dicho valor se le llama **incremento.** La forma más general de la instrucción **FOR ... TO** es

```
FOR K = A TO B STEP C
```

En este caso, el contador **K** avanza a partir de su valor inicial **A** en incrementos de valor **C** hasta que alcanza o sobrepasa el valor final **B.** Si **B** es menor que **A**, **C** puede ser negativo.

5.5

Bucles anidados

En algunos casos es necesario crear un bucle dentro de otro, y a eso se le llama **bucle anidado.** De hecho, esta operación puede extenderse a más de dos niveles, con la sola limitación de que los diversos bucles no solapen unos con otros; los bucles interiores deben estar completamente contenidos en los exteriores.

Las instrucciones que controlan dos bucles internos son las siguientes:

```
FOR I = 1 TO N
FOR J = 1 TO M

NEXT J
NEXT I
```

Observe que el bucle interior acaba antes que el exterior. En este ejemplo, el bucle interior se repite **N** × **M** veces.

5.6

Programa ejemplo 5.2

Este programa sirve para calcular el número de horas que hay que pagar a los obreros de una fábrica. Para cada uno se introducen un número de trabajos y una serie de cinco horas diferentes de inicio y fin de trabajo. Se calcula el tiempo de trabajo de cada día, que se suma al total semanal. El resultado para cada obrero es un número de trabajos y un total de horas de trabajo.

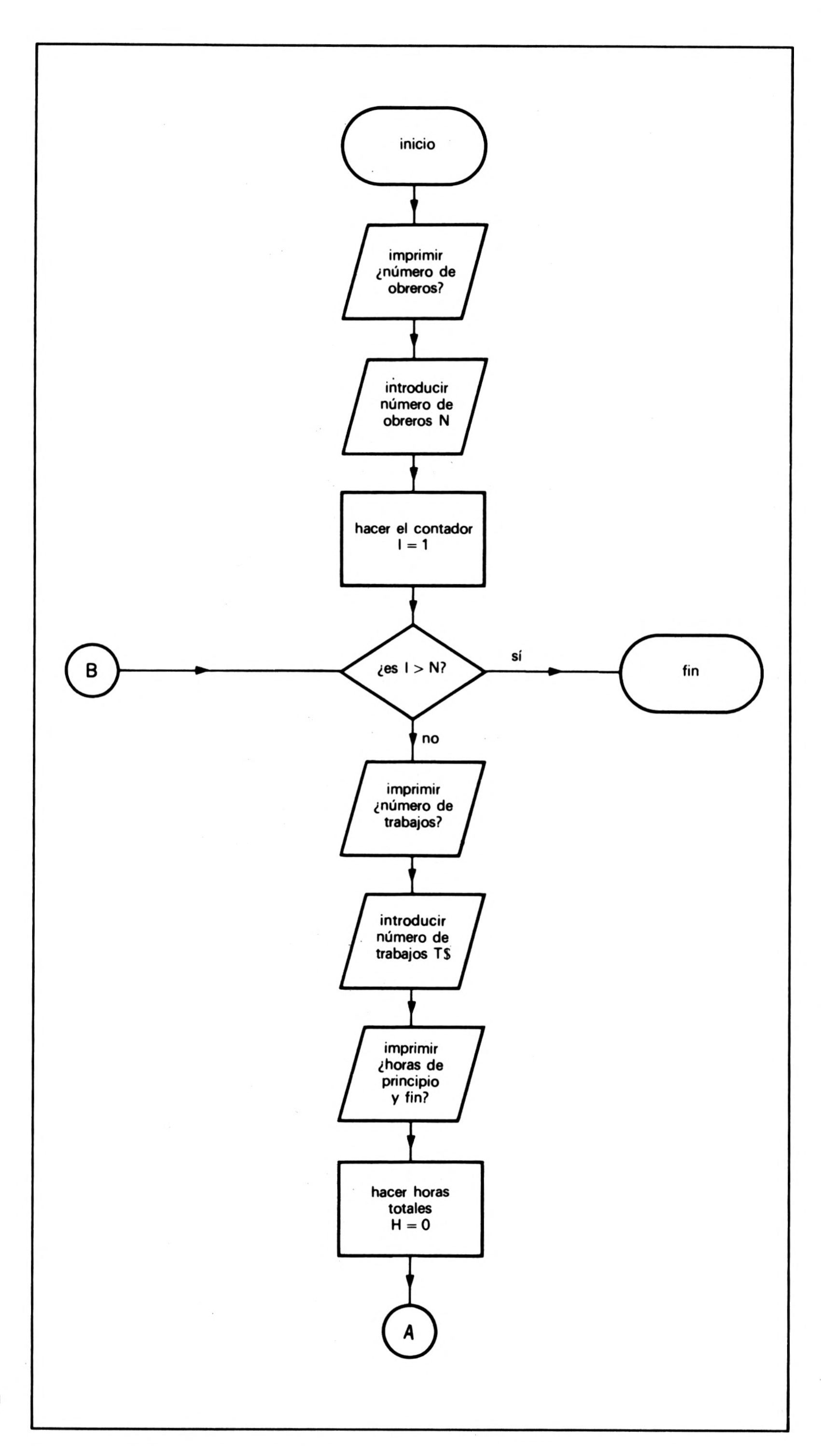
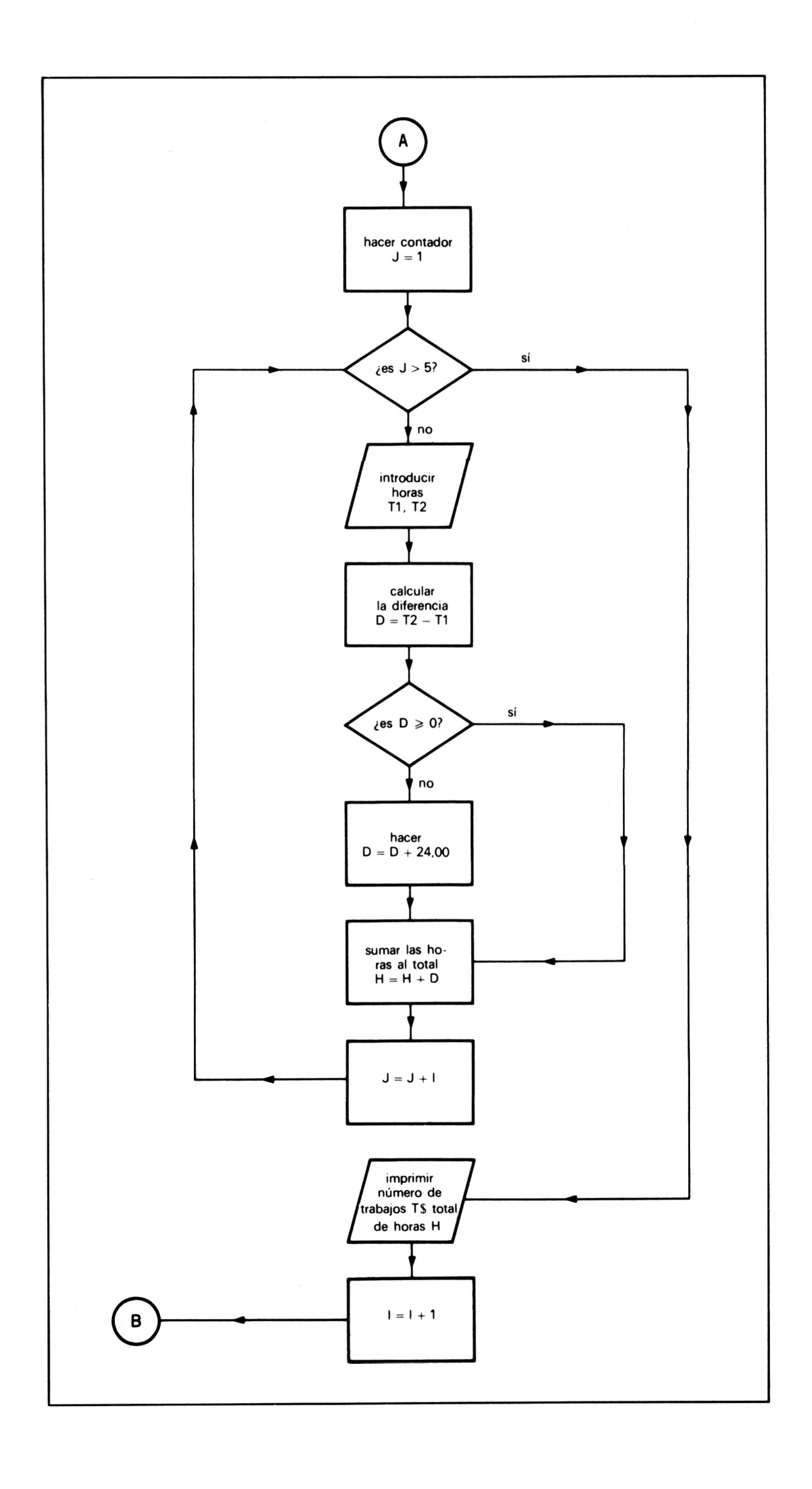


Figura 5.3
Diagrama de flujo del programa ejemplo 5.2


A
hacer contador
J = 1
¿es J > 5?
sí
no
introducir
horas
T1, T2
calcular
la diferencia
D = T2 − T1
¿es D ⩾ 0?
sí
no
hacer
D = D + 24,00
sumar las ho-
ras al total
H = H + D
J = J + I
imprimir
número de
trabajos T$ total
de horas H
I = I + 1
B

Método

El algoritmo utilizado para redactar el programa es el siguiente:

Introducir el número de obreros.

Para cada obrero, repetir el siguiente proceso:

Introducir el número de trabajos.
Poner el total de horas a cero.
Para cada día, repetir el siguiente proceso.

Introducir las horas de comienzo y final del trabajo.
Calcular el tiempo trabajado.
Añadir el resultado al total.

Presentar como resultado el número de trabajos y el total.

Los tiempos se expresan en horas sobre una base de 24 horas y en décimas de hora. Así, 1.730 horas se expresa 17,30. Si una diferencia horaria es negativa por haberse ejecutado el trabajo desde antes de la medianoche hasta pasada la misma, se suma 24,00 para corregir el valor.

Variables

N Número de obreros.
T1 Tiempo de inicio.
T2 Tiempo de terminación.
T$ Número de trabajos:
D Horas trabajadas en un día.
H Total de horas trabajadas en una semana.
I, J Contadores.

Programa

```
100 REM PROGRAMA EJEMPLO 5.2
105 REM HORAS DE TRABAJO SEMANAL
110 REM
115 PRINT "NUMERO DE OBREROS?";
120 INPUT N
125 FOR I = 1 TO N
130 PRINT "NUMERO DE TRABAJOS?";
135 INPUT T
140 PRINT "TIEMPOS DE INICIO Y TERMINACION?"
145 LET H = 0
150 FOR J = 1 TO 5
155 INPUT "T1"; T1:INPUT "T2"; T2
160 LET D = T2 - T1
```

```
165 IF D >= 0 THEN 175
170 LET D = D + 24.00
175 LET H = H + D
180 NEXT J
185 PRINT "NUMERO DE TRABAJOS "; T
190 PRINT "TOTAL DE HORAS "; H
195 NEXT I
200 END
```

Puntos de interés

- Observe cuidadosamente las porciones del programa situadas en el bucle interno, las situadas en el externo y las que no pertenecen a ninguno de los dos.
- La línea 165 recoge una condición de no acción.
- En algunas versiones de ejecución práctica del Basic, la entrada por teclado se solicita mediante un signo de interrogación. En ese caso, aparecería un signo de interrogación en los contenidos de pantalla correspondientes a las líneas 115, 130 y 140.

5.7

Bifurcaciones de entrada y salida de un bucle

Una vez escrita la parte del programa correspondiente a un bucle, es muy tentador aprovechar algunas o todas las instrucciones contenidas en esa parte para otros fines, lo que obliga a utilizar bifurcaciones para acceder al bucle desde otros puntos del programa.

Esa tentación debe resistirse a toda costa. Ramificar un programa hacia el interior de un bucle es una costumbre completamente desaconsejable que, en la mayor parte de los ordenadores, tiene como consecuencia la detención del programa en una condición de error.

Otra cosa es crear una bifurcación para salir de un bucle, situación que se plantea cuando la terminación del mismo depende de más de una sola condición. Una de ellas es la llegada del contador a su valor final, y la otra supone alguna verificación realizada dentro del propio bucle.

Los diversos tipos de ordenador responden de diferente forma a la instrucción de ramificación que transfiere el control fuera del bucle. En unos se mantiene el valor real del contador, que en otros se pierde; los hay que conservan el valor del contador y el límite de repeticiones del bucle hasta el final del programa y reducen la velocidad de procesado del mismo.

Por ello no es recomendable saltar directamente al exterior de un

bucle. Lo mejor es pasar el contador a su valor final y saltar hasta el fin del bucle, técnica que queda ilustrada en el siguiente programa ejemplo.

Queda todavía por hacer una última observación respecto a los bucles controlados por instrucciones **FOR ... TO** y **NEXT**, y es que el límite del bucle (el valor final del contador) no puede modificarse desde el interior de dicho bucle. En la mayor parte de los ordenadores, los intentos de efectuar esa manipulación no producen ningún efecto.

5.8 Programa ejemplo 5.3

Este es una versión mejorada del programa de cálculo de la medida del ejemplo 5.1. Si el usuario observa que se ha producido un error durante la introducción de los números, introduce a su vez un valor "ficticio" (cero en este caso) que termina el bucle de entrada y permite al usuario volver a empezar.

Método

El programa revisado utiliza una variable de control que se pone a cero antes de que arranque el bucle.

Dentro de dicho bucle, si se detecta la introducción de un valor cero,

la variable de verificación pasa a 1,
se presenta un mensaje,
el contador del bucle pasa a su valor final,
el control se transfiere al final del bucle.

Tras el bucle, si la variable de control vale 1, aquél se repite.

Variables

Las del ejemplo 5.1, más

C Variable de control.

Diagrama de flujo

Se ilustra en la figura 5.4.

Programa

```
100 REM PROGRAMA EJEMPLO 5.3
105 REM INTRODUCCION Y CALCULO DE LA MEDIA DE
        UNA SERIE DE NUMEROS
110 REM VERSION MEJORADA DEL PROGRAMA EJEMPLO
        5.1
115 REM
120 PRINT "DE CUANTOS NUMEROS SE VA A CALCULAR
          LA MEDIA?"
125 INPUT N
130 PRINT "INTRODUZCA LOS NUMEROS"
135 LET C = 0
140 LET T = 0
```

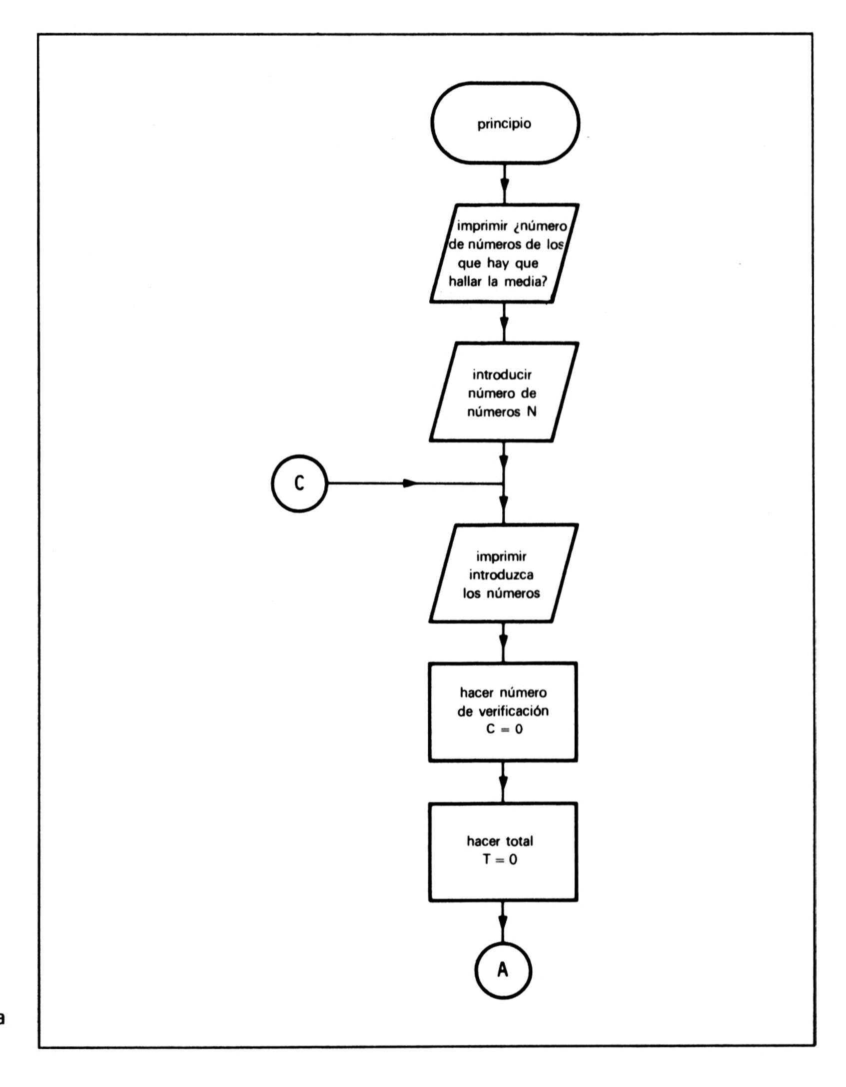
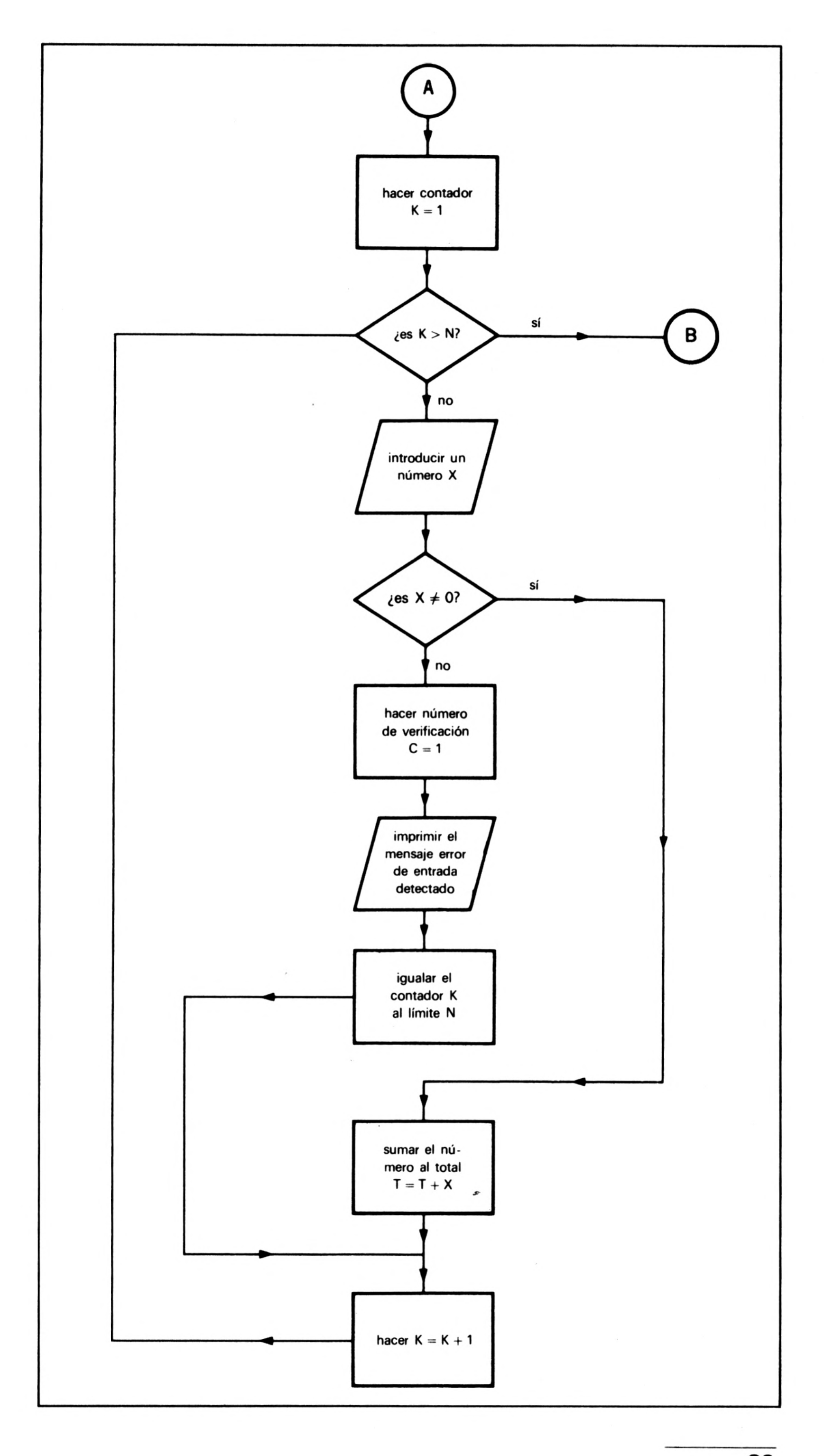
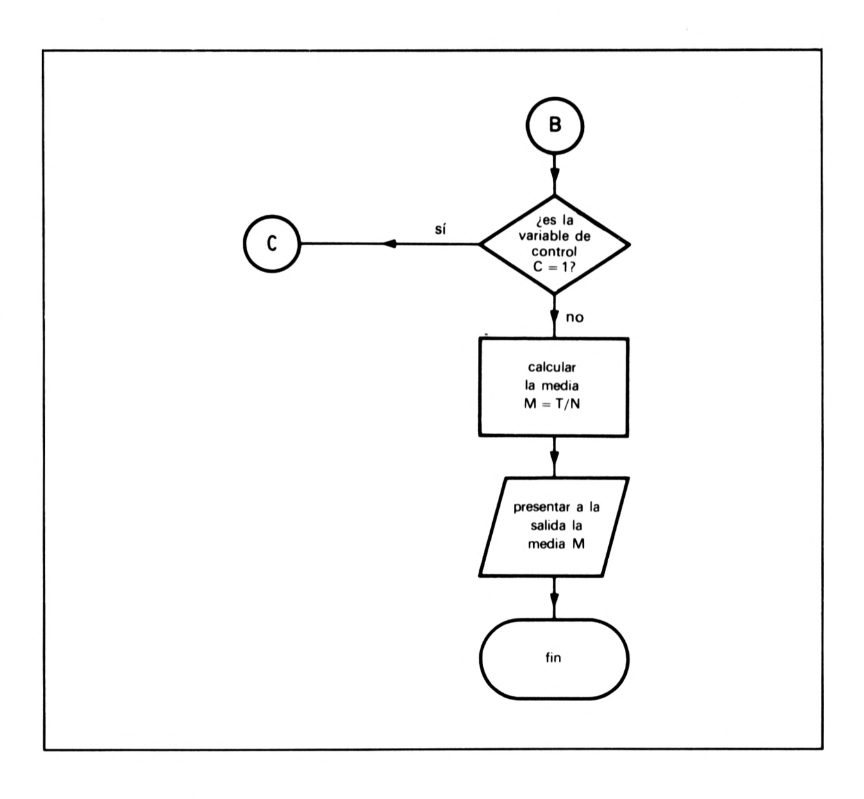


Figura 5.4
Diagrama de flujo del programa ejemplo 5.3


A
hacer contador
K = 1
¿es K > N?
sí
B
no
introducir un
número X
¿es X ≠ 0?
sí
no
hacer número
de verificación
C = 1
imprimir el
mensaje error
de entrada
detectado
igualar el
contador K
al límite N
sumar el nú-
mero al total
T = T + X
hacer K = K + 1




```
145 FOR K = 1 TO N
150 INPUT X
155 IF X <> 0  THEN 180
160 LET C = 1
165 PRINT "ERROR DETECTADO EN LA ENTRADA DE
          NUMEROS"
170 LET K = N
175 GOTO 185
180 LET T = T + X
185 NEXT K
190 IF C = 1 THEN 130
195 LET M = T/N
200 PRINT "MEDIA"; M
205 END
```

Puntos de interés

- Si el control vuelve al inicio del bucle, la variable de control **C** debe ponerse de nuevo a cero, porque en caso contrario el programa no terminaría nunca.
- Dentro del bucle, el contador **K** se actualiza hasta su valor límite **N**. **El valor de N no se altera.**

5.9

Repetición mientras se cumple cierta condición

Pasemos ahora a otra forma de controlar un bucle: la repetición del mismo *mientras* se cumpla cierta condición. El diagrama de flujo de la figura 5.5 ilustra la estructura lógica de esta clase de bucle. Observe que la condición se verifica *antes* de cada nueva repetición, lo que permite que el bucle ni siquiera llegue a ejecutarse una sola vez.

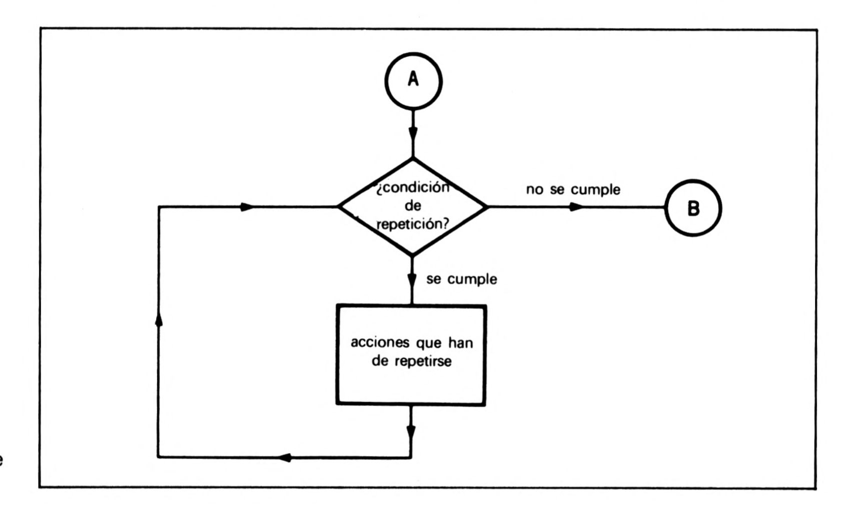


Figura 5.5
Repetición mientras se cumple cierta condición

5.10

Programa ejemplo 5.4

Este segmento de programa supervisa la introducción de una serie de nombres, direcciones y números de teléfono. El final de los datos queda indicado por los caracteres *****, conocidos como **indicador de fin de datos** o **valor parásito.** El segmento de programa recogido aquí se limita a reflejar la entrada, es decir, a presentar en pantalla los datos introducidos. En la práctica, éstos quedarían almacenados para su ulterior procesado en otra parte del programa.

Método

El siguiente algoritmo indica el método de trabajo del segmento de programa del ejemplo:

Se introduce un nombre.

Si ese nombre no es el indicador de fin de los datos, se repite el siguiente proceso:

Se introducen una dirección y un número de teléfono.
Se presentan en pantalla nombre, dirección y número de teléfono.
Se introduce un nombre.

Variables

N$ Nombre.
D$ Dirección.
T$ Número de teléfono.

Diagrama de flujo

Se ilustra en la figura 5.6.

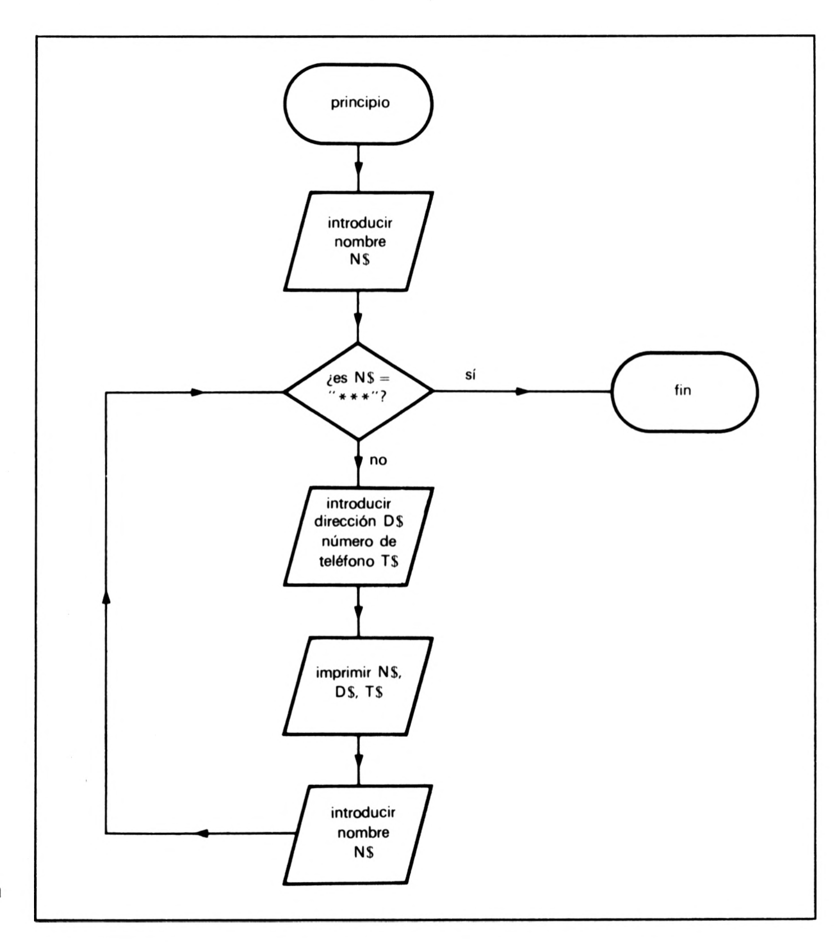


Figura 5.6
Diagrama de flujo del programa ejemplo 5.4

Programa

```
100 REM PROGRAMA EJEMPLO 5.4
105 REM ENTRADA DE NOMBRES, DIRECCIONES, NUMEROS
        DE TELEFONO
110 REM
115 INPUT N$
120 IF N$ = "***" THEN 145
125 INPUT D$, T$
130 PRINT N$, D$, T$
135 INPUT N$
140 GOTO 120
145 END
```

Puntos de interés

- El bucle va de la línea 120 a la 140.
- La condición utilizada no es una condición para continuar el bucle, sino para terminarlo.

5.11 Repetición hasta que se cumpla cierta condición

La tercera y última técnica de control de un bucle —es decir, su repetición *hasta que* se cumpla cierta condición— queda representada en la figura 5.7 en su estructura lógica. Observe que la condición

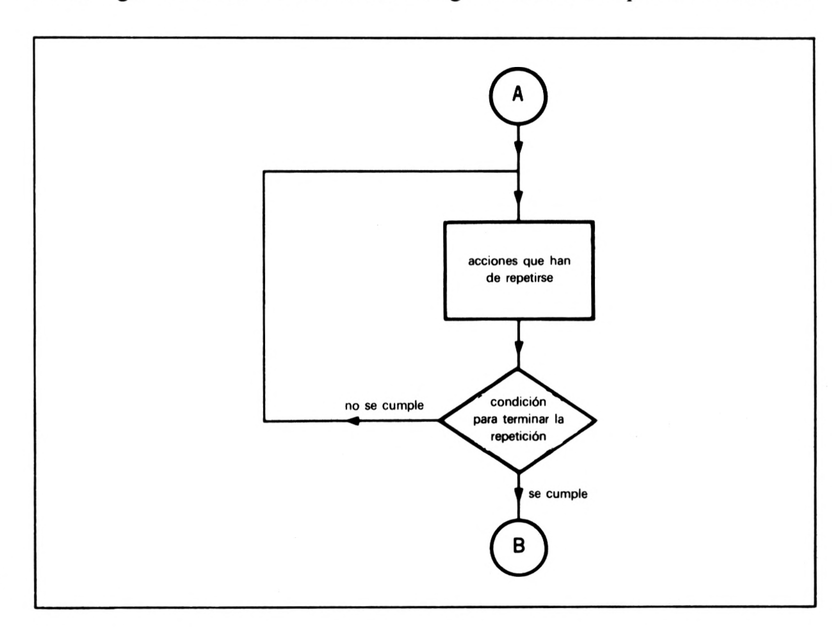


Figura 5.7
Repetición hasta que se cumpla cierta condición

aparece *al final* del bucle que, por tanto, se ejecuta siempre al menos una vez.

5.12
Programa ejemplo 5.5

Si la población de una especie determinada crece a razón de cierto porcentaje anual, ¿tras cuántos años se duplicará su tamaño original?

La siguiente fórmula permite determinar aproximadamente el incremento de la población:

$$P = P_0\left(1 + \frac{R}{100}\right)^T$$

P: Población actual
P_0: Población original
R: Porcentaje de incremento anual
T: Número de años

Para que la población duplique su tamaño original, la relación

$$\frac{P}{P_0} = 2,$$

es decir, $\left(1 + \frac{R}{100}\right)^T = 2.$

Método

Se introduce el porcentaje de incremento R y se verifica que sea positivo. El valor de T se pone a cero.

Se repiten los pasos indicados a continuación hasta que $\left(1 + \frac{R}{100}\right)^T$ supera el valor de 2:

T aumenta una unidad.

Se calcula el valor de $\left(1 + \frac{R}{100}\right)^T$.

Se presenta como resultado el valor actual de T.

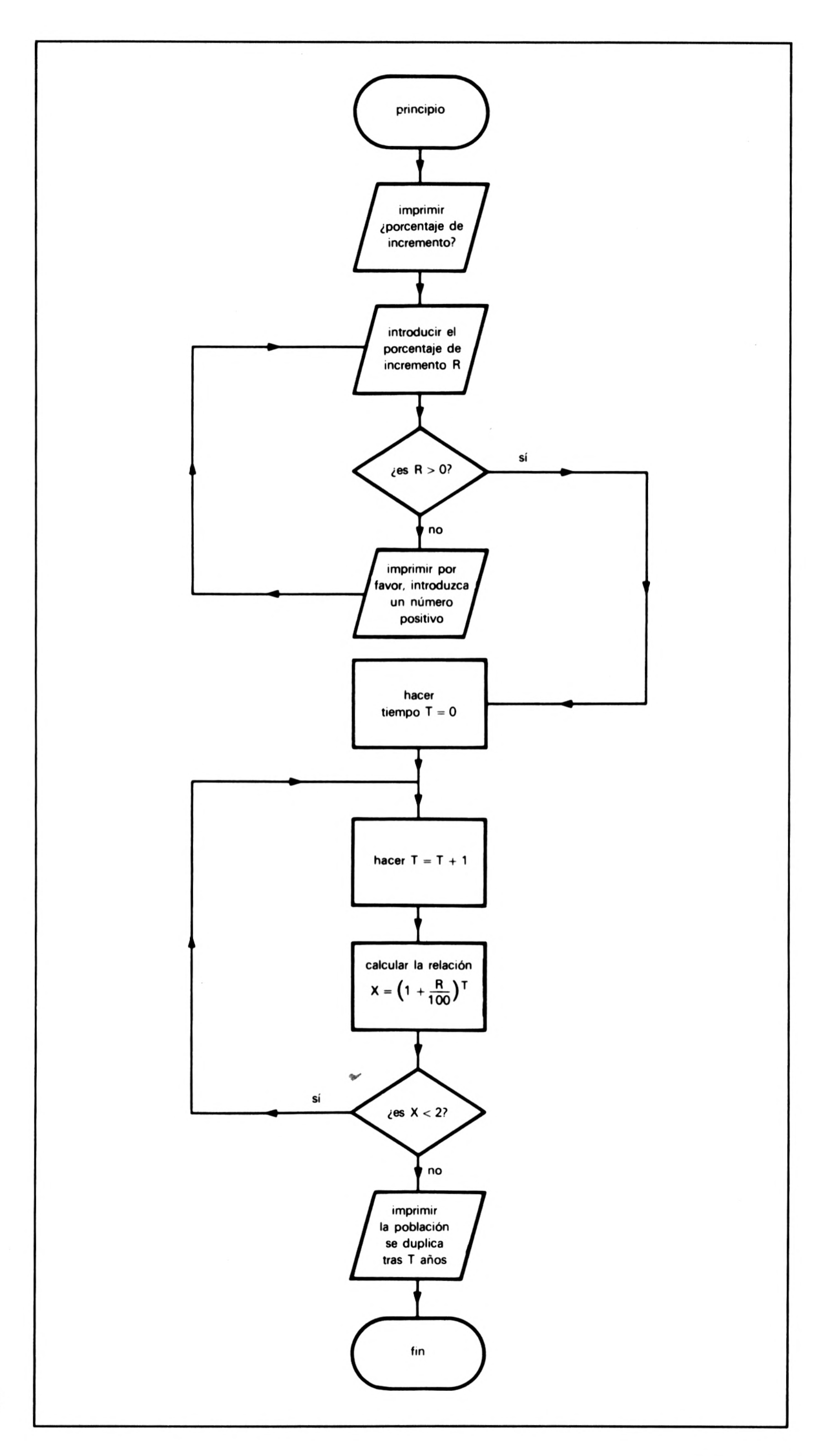


Figura 5.8
Diagrama de flujo del programa ejemplo 5.5

Variables

R Porcentaje de incremento.
T Tiempo (en años).
X Relación de poblaciones.

Diagrama de flujo

Véase figura 5.8.

Programa

```
100 REM PROGRAMA EJEMPLO 5.5
105 REM CALCULOS DE POBLACION
110 REM
115 PRINT "PORCENTAJE DE INCREMENTO?"
120 INPUT R
125 IF R > 0 THEN 140
130 PRINT "POR FAVOR INTRODUZCA UN NUMERO POSITI
         VO";
135 GOTO 120
140 LET T = 0
145 LET T = T + 1
150 LET X = (1 + R/100 )^ T
155 IF X < 2 THEN 145
160 PRINT "LA POBLACION SE DUPLICA DESPUES DE";
165 PRINT T; "ANOS"
170 END
```

Puntos de interés

- El bucle va de la línea 145 a la 155.
- La condición expresada en la línea 155 es de continuación del bucle.
- La condición de la línea 125 sirve para evitar las dos líneas siguientes.
- El método que acaba de proponerse no es, ni mucho menos, la forma más eficaz de calcular la relación **X**. En el ejercicio incluido al final de este capítulo se discute un enfoque mejor.

5.13 Conclusión

En este capítulo hemos introducido el concepto de bucle de programa, junto con tres formas de estructurarlo. Estas son:

- Repetición mientras se cumple cierta condición.
- Repetición hasta que se cumpla cierta condición.
- Repetición durante determinado número de veces.

Al redactar un bucle, es importante tener muy claro qué se pretende conseguir con ello y de qué forma conviene controlar la repetición. Si hay varias condiciones de terminación, es aconsejable reunirlas todas en una sola instrucción de bifurcación.

Por último, hay que recordar que algunas operaciones referidas a los bucles se ejecutan de diferente forma en unos ordenadores que en otros.

5

Ejercicio

1. Resuma las características diferenciales de las tres formas de terminar un bucle.
2. La forma más general de la instrucción **FOR ... TO** es

```
FOR K = A TO B STEP C
```

   a) Modifique el diagrama de flujo de la figura 5.1 para adaptarlo a esta forma general.
   Indique en cada uno de los siguientes casos el número de veces que se repetirá el bucle:
   b) **A** = 1, **B** = 20, **C** = 1
   c) **A** = 1, **B** = **N**, **C** = 1
   d) **A** = 1, **B** = 10, **C** = 0,5
   e) **A** = 3, **B** = −3, **C** = −1
   f) **A** = 1, **B** = 10, **C** = 2
   g) **A** = **X**, **B** = **Y**, **C** = **Z**
3. Escriba un segmento de programa para controlar la introducción de diez series de nombres de lugares y cifras de temperaturas y precipitaciones.
4. El siguiente segmento de programa se ha creado con el fin de controlar la introducción de una serie de veinte números, pero, además, puede interrumpirse antes de tiempo mediante la introducción del indicador de fin de los datos **0**.

```
100 LET N = 20
105 FOR C = 1 TO N
110 INPUT X
115 IF X = 0 THEN 130
120 REM SE HAN OMITIDO LAS SENTENCIAS PARA
        PROCESAR X
125 GOTO 135
130 LET N = C
135 NEXT C
```

   Corrija el error que contiene.

5. El siguiente segmento de programa contiene un bucle interior a otro:

```
500 FOR I = 1 TO 5
505 REM BUCLE EXTERNO
510 FOR J = 1 TO 8
515 REM BUCLE INTERNO
520 NEXT I
525 NEXT J
```

   a) Corrija el error que contiene.
   b) ¿Cuántas veces se repetiría el bucle interno en el segmento corregido?

6. Modifique el programa ejemplo 5.1 con el fin de incluir un método para verificar que la cantidad de números (variable **N**) sea positivo.
7. Suponiendo que no dispone de la función raíz cuadrada, escriba un programa para determinar el menor entero positivo cuyo cuadrado es superior a 100.000.
8. El programa siguiente es una versión alternativa del presentado en el ejemplo 5.4.

```
100 REM PROGRAMA EJEMPLO 5.4, VERSION
        ALTERNATIVA
105 REM ENTRADA DE NOMBRES, DIRECCIONES, NUMEROS
        DE TELEFONO
110 REM
115 INPUT N$,D$,T$
120 IF N$ = "***" THEN 135
125 PRINT N$, D$, T$
130 GOTO 115
135 END
```

   Exponga claramente las similitudes y diferencias que hay entre esta versión y la anterior, tanto desde el punto de vista lógico como desde el punto de vista del usuario.

9. El programa ejemplo 5.5 constituye una forma bastante ineficaz de resolver el problema propuesto. El fallo está en la línea 150, en la que se calcula la relación **X** elevando una cantidad a la potencia **T**, operación que el ordenador ejecuta con mucha lentitud. Como el valor actual de **T** es superior en una unidad al anterior, el de **X** es (1 + **R**/100) veces el anterior. Utilice esta información y un valor inicial de **X** adecuado para modificar esta porción del programa de forma tal que no sea necesaria la elevación a la potencia **T**.
10. Escriba un programa que satisfaga las especificaciones indicadas a continuación:

   El programa permitirá utilizar un ordenador como guía telefónica personal, para lo que han de almacenarse una serie de nombres y números de teléfono. Para usar el programa, se introduce el nombre; esta operación abre el acceso a la lista de nombres almacenada, y si en ella figura alguno igual al introducido, el ordenador presenta como resultado el número de teléfono correspondiente. En caso contrario, presenta un mensaje adecuado.

   El programa ha de escribirse de manera que sea fácil añadir a la lista memorizada más nombres y números. Además, debe bastar un solo pase del mismo para averiguar los números de teléfono correspondientes a varios nombres.

   Sugerencias: la mejor forma de almacenar los nombres y números es

mediante instrucciones **DATA**. Para acceder a ellos se emplea una instrucción **READ**. La instrucción **RESTORE** permite acceder de nuevo a los datos para efectuar otra consulta.

11. Escriba un programa capaz de producir una tabla de cuadrados, cubos, raíces cuadradas e inversos de todos los enteros comprendidos entre 1 y 100.

12. Escriba un programa que haga multiplicaciones de enteros mediante una serie de adiciones repetidas o divisiones de enteros por sustracciones repetidas.


RESET
RETURN

6
Manipulación de caracteres

Los dispositivos de manipulación de caracteres alfanuméricos con que cuenta el Basic están entre las características más potentes de ese lenguaje. Este capítulo va dedicado precisamente a describir esos dispositivos, presentes en la mayor parte de las versiones, que se ilustran con algunos ejemplos sobre su utilización.

Constituye este capítulo una etapa esencial en el dominio de la programación en Basic a su nivel elemental, porque aprovecha la mayor parte del material estudiado hasta el momento y será de aplicación en casi todos los capítulos que le siguen.

6.1 Dispositivos de manipulación de cadenas de caracteres en Basic

La figura 6.1 recoge los dispositivos de manipulación de cadenas de caracteres del lenguaje Basic de referencia. Incluyen un **operador,** que reúne dos cadenas de caracteres y produce otra como resultado, y varias **funciones** que aceptan uno o más **argumentos** y producen como resultado un valor único. Los términos función y argumento se explican en la sección 6.2.

La brevedad de la lista es intencionada, porque se trataba de incluir en ella sólo los dispositivos esenciales e independientes del tipo de ordenador. De todas formas, el conjunto presentado constituye una herramienta de manipulación de caracteres muy poderosa. En las próximas secciones se explicarán con más detalle las características de los recursos contenidos en la figura 6.1.

6.2

Funciones y argumentos

Antes de discutir las funciones de manipulación de cadenas, conviene dedicar unas pocas palabras al concepto de función y a los de argumento y valor asociados al mismo. Las ideas presentadas aquí se discutirán más detalladamente en el capítulo 8.

Una **función** transforma una cantidad en otra. Así, la función raíz cuadrada transforma el número 25 en el número 5.

La cantidad sobre la que opera la función se llama **argumento** de la función; en el ejemplo anterior, el argumento es 25. Habitualmente, el argumento se escribe entre paréntesis a continuación del nombre de la función. Así, el argumento de **SQR(X)** es **X**.

Operador

+	une cadenas alfanuméricas

Funciones

ASC(X$)	Obtiene el número correspondiente al código ASCII para el primer carácter de X$.
CHR$(X)	Obtiene el carácter cuyo código ASCII es el número X.
LEN(X$)	Obtiene el número de caracteres de la cadena X$.
MID$(X$,A,B)	Obtiene la subcadena de X$ de longitud B caracteres que empieza en el carácter A.
STR$(X)	Obtiene la cadena alfanumérica equivalente al número X.
VAL (X$)	Obtiene el equivalente numérico a la cadena alfanumérica X$.

Nota

En algunas versiones del Basic:

CHR(X$) sustituye a ASC(X$)
SUB$(X$,A,B) sustituye a MID$(X$,A,B)
& sustituye a +

Figura 6.1
Recursos de manipulación de caracteres del lenguaje de referencia

El resultado producido por una función es el **valor** de la misma. En el ejemplo que venimos utilizando, el valor de **SQR(X)** es 5.

6.3

Operador de unión +

El operador de unión + combina dos cadenas de caracteres y los convierte en una cadena única. Por ejemplo, dados **A$** = "**RATON**" y **B$** = "**ERA**", la instrucción

```
LET C$ = A$ + B$
```

asigna a **C$** el valor "**RATONERA**".

6.4

ASC(X$)

La función **ASC** produce como resultado el número ASCII (Código Normalizado Estadounidense para el Intercambio de Información) que corresponde al primero de los caracteres de la cadena que sirve de argumento. Por ejemplo, si **X$** = "**MAS**", el valor de **ASC(X$)** es 77, código ASCII correspondiente a la letra M (véase figura 6.2).

La función ASC se emplea frecuentemente en operaciones condicionales, como ilustra este ejemplo: la condición para que una variable **X$** sea una letra mayúscula es que su código ASCII esté entre 65 y 90, ambos inclusive (véase figura 6.2). La instrucción

```
IF ASC(X$) >= 65 AND ASC(X$) <= 90
```

expresa la condición mencionada.

6.5

CHR$(X)

La función **CHR$** produce como resultado el carácter que en el código ASCII corresponde al argumento. Por ejemplo, si **X** = 33, **CHR$(X)** toma el valor "!" (véase figura 6.2).

El argumento de **CHR$** ha de ser un entero comprendido entre 0 y 127. Si no se cumple esa condición, el resultado de la función varía de

Carácter	Código (decimal)	Carácter	Código (decimal)	Carácter	Código (decimal)
(espacio)	32	@	64	`	96
!	33	A	65	a	97
"	34	B	66	b	98
#	35	C	67	c	99
$	36	D	68	d	100
%	37	E	69	e	101
&	38	F	70	f	102
'	39	G	71	g	103
(	40	H	72	h	104
)	41	I	73	i	105
*	42	J	74	j	106
+	43	K	75	k	107
,	44	L	76	l	108
—	45	M	77	m	109
.	46	N	78	n	110
/	47	O	79	o	111
0	48	P	80	p	112
1	49	Q	81	q	113
2	50	R	82	r	114
3	51	S	83	s	115
4	52	T	84	t	116
5	53	U	85	u	117
6	54	V	86	v	118
7	55	W	87	w	119
8	56	X	88	x	120
9	57	Y	89	y	121
:	58	Z	90	z	122
;	59	[	91	{	123
<	60	\	92		124
=	61	]	93	}	125
>	62	^	94	~	126
?	63	_	95		

Figura 6.2
Códigos de algunos caracteres ASCII de uso frecuente

un ordenador a otro, aunque en la mayor parte de ellos aparece una indicación de error.

La función **CHR$** es la **inversa** de la ASC, lo que significa que actúa en la dirección contraria. Para cualquier valor de **X** comprendido entre 0 y 127,

```
ASC(CHR$(X)) = X
```

Por ejemplo, si **X** = 100, **CHR$(X) = d**, y **ASC(d)** = **100** (véase figura 6.2).

6.6

LEN(X$)

La función **LEN** produce como resultado el número de caracteres que contiene su argumento. Por ejemplo, si **X$** = "**ABCDEF**",

LEN(X$) toma el valor 6. Si **X$** es la **serie nula,** es decir, " ", **LEN(X$) = 0**.

6.7 MID$(X$, A, B)

MID$ es una función de **extracción** que produce como resultado una subcadena de la cadena **X$** con **B** caracteres de longitud contados a partir del carácter que ocupa el lugar **A**. Por ejemplo, si **X$** = "**AHORA ES EL MOMENTO**", **MID$(X$, 7, 2)** tiene el valor "**ES**".

La posición de partida y la longitud de la subcadena deben elegirse de manera que no ocurra parcial o totalmente fuera de la cadena. Así, en el ejemplo anterior, **MID$(X$, 20, 2)** está fuera de **X$**. Para ello, los valores de los argumentos **A** y **B** deben satisfacer la condición

```
A + B <= LEN(X$) + 1
```

Algunos ordenadores toleran la transgresión de esta norma, pero es un vicio de programación que debe evitarse a toda costa.

6.8 STR$(X)

La función **STR$** produce como resultado una cadena alfanumérica equivalente al número que le sirve de argumento. Por ejemplo, si **X** vale 99, **STR$(X)** produce como resultado la serie "**99**". En otras palabras: **STR$** transforma una variable numérica en otra alfanumérica. Se utiliza, por ejemplo, cuando el resultado de un cálculo o del contador de un bucle debe pasar a formar parte de una variable alfanumérica.

Así, si **D** es el día, **M** el mes y **A** el año, la instrucción

```
LET F$ = STR$(D) + "/" + STR$(M) + "/" + STR$(A)
```

asigna a la variable **F$** una fecha escrita en la forma "**22/08/49**".

6.9

VAL(X$)

La función **VAL** produce como resultado el valor numérico de la cadena que le sirve de argumento. Por ejemplo, si el valor de **X$** es "**2001**", **VAL(X$)** es 2001. En otras palabras: **VAL** transforma un valor alfanumérico en otro numérico. Se emplea, por ejemplo, para extraer la porción numérica de una cadena y utilizarla en cálculos ulteriores.

Así, si **F$** es la fecha "**01/04/84**", la instrucción

```
LET A = VAL(MID$(D$, 7, 2))
```

asigna el valor 84 a la variable **A**. Naturalmente, el argumento de **VAL** ha de ser una cadena de caracteres numéricos; en caso contrario, se producirá un error.

La función **VAL** es inversa de la **STR$**. En otras palabras:

VAL(STR$(X)) es **X**.

6.10

Programas ejemplo

Discutiremos ahora dos programas que darán idea de las diversas aplicaciones de las funciones de manipulación de cadenas. Casi todos los programas contenidos en el libro a partir de este momento harán uso de las mencionadas funciones.

6.11

Programa ejemplo 6.1

El objetivo de este programa es localizar una subcadena dentro de una cadena. Dadas una cadena y una subcadena, indica la posición en la primera del carácter inicial de la segunda, o da un valor 0 si la subcadena no se encuentra en la cadena.

Por ejemplo, dada la cadena "**NO POR MUCHO MADRUGAR AMANECE MAS TEMPRANO**" y la subcadena "**MADRUGAR**", el programa dará como resultado el número 14, posición de partida de "**MADRUGAR**" dentro de la cadena. Además, está escrito de manera que en otra ocasión pueda pasar a formar parte de otro programa más amplio.

Método

El algoritmo seguido es:

Determinar la longitud **L1** de la cadena y la longitud **L2** de la subcadena.
Poner a 0 la posición inicial **P** de la subcadena dentro de la serie.
Para cada posible posición inicial **I** desde 1 hasta **L1** – **L2** + 1, repetir el proceso.

Si la cadena de caracteres de longitud **L2** que empieza en la posición **I** coincide con la subcadena,

hacer **P** igual a **I**

terminar la repetición.

La interpretación de este algoritmo en Basic supone el uso de un bucle **FOR ... TO** con una condición de finalización contenida en su interior. Se emplea el método de bifurcación de salida de un bucle descrito en el capítulo anterior. La forma más eficaz de expresar la condición es escribirla como condición de que el conjunto de caracteres no coincida con la cadena.

Variables

A$ Cadena.
B$ Subcadena.
L1 Longitud de la cadena.
L2 Longitud de la subcadena.
P Posición de partida de la subcadena dentro de la cadena.
I Contador del bucle.

Diagrama de flujo

La figura 6.3 ilustra el flujo de control del programa.

Programa

```
5000 REM PROGRAMA EJEMPLO 6.1
5005 REM LOCALIZACION DE UNA SUBCADENA EN UNA
         CADENA
5010 REM
5015 LET L1 = LEN(A$)
5020 LET L2 = LEN(B$)
```

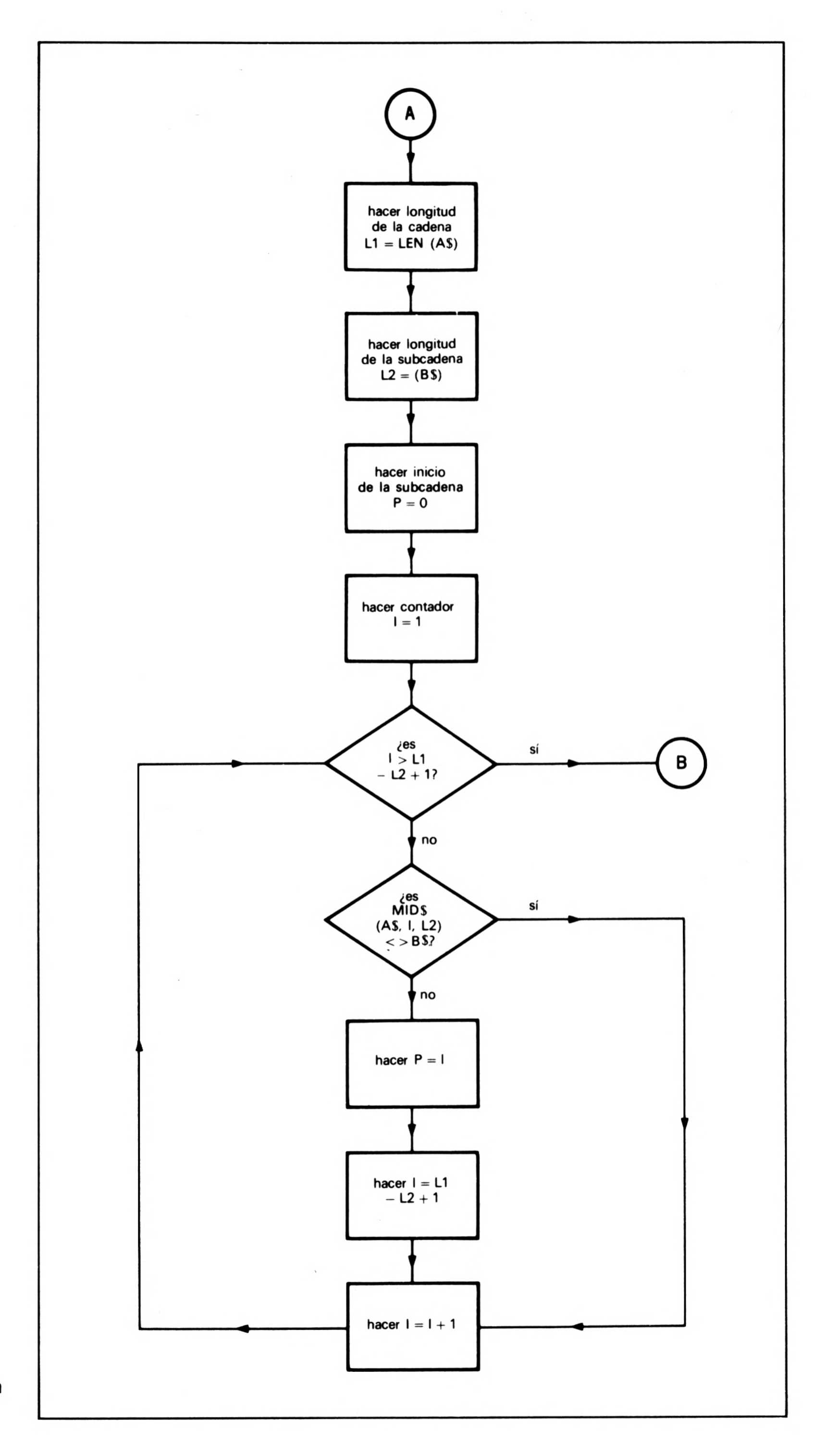


Figura 6.3
Diagrama de flujo del programa ejemplo 6.1

```
5025 LET P = 0
5030 FOR I = 1 TO L1 - L2 + 1
5035 IF MID$(A$,I,L2) <> B$ THEN 5050
5040 LET P = I
5045 LET I = L1 - L2 + 1
5050 NEXT I
5055 REM
```

Puntos de interés

- No es un programa completo, porque carece de los segmentos de entrada y salida.
- El programa tiene en cuenta la posibilidad de que se presenten casos como **L2** > **L1** (en el que el bucle no se ejecuta) o como la presencia de una cadena o una subcadena nulas.

6.12 Programa ejemplo 6.2

Dadas dos fechas, este programa calcula el número de días que las separan. La resolución del problema es mucho más difícil de lo que parece a primera vista, y en la programación comercial se dedica mucho esfuerzo a la manipulación eficaz de las fechas.

El método propuesto es aproximado, porque no tiene en cuenta los años bisiestos, pero tiene muchas aplicaciones prácticas, como la determinación de las fechas de vencimiento de los pagos o el retraso de éstos. El programa está escrito de manera que pueda pasar a integrarse en otro más amplio, y carece de instrucciones de entrada y salida.

Método

Las fechas se almacenan en la forma día/mes/año (01/01/80, por ejemplo). Para calcular la diferencia entre dos, se expresan como número de días desde el comienzo de siglo según el siguiente procedimiento:

El año se multiplica por 365.

El mes se usa para entrar en una tabla que contiene los días del año transcurridos hasta el comienzo del mes (0 para enero, 31 para febrero, 59 para marzo, etc.).

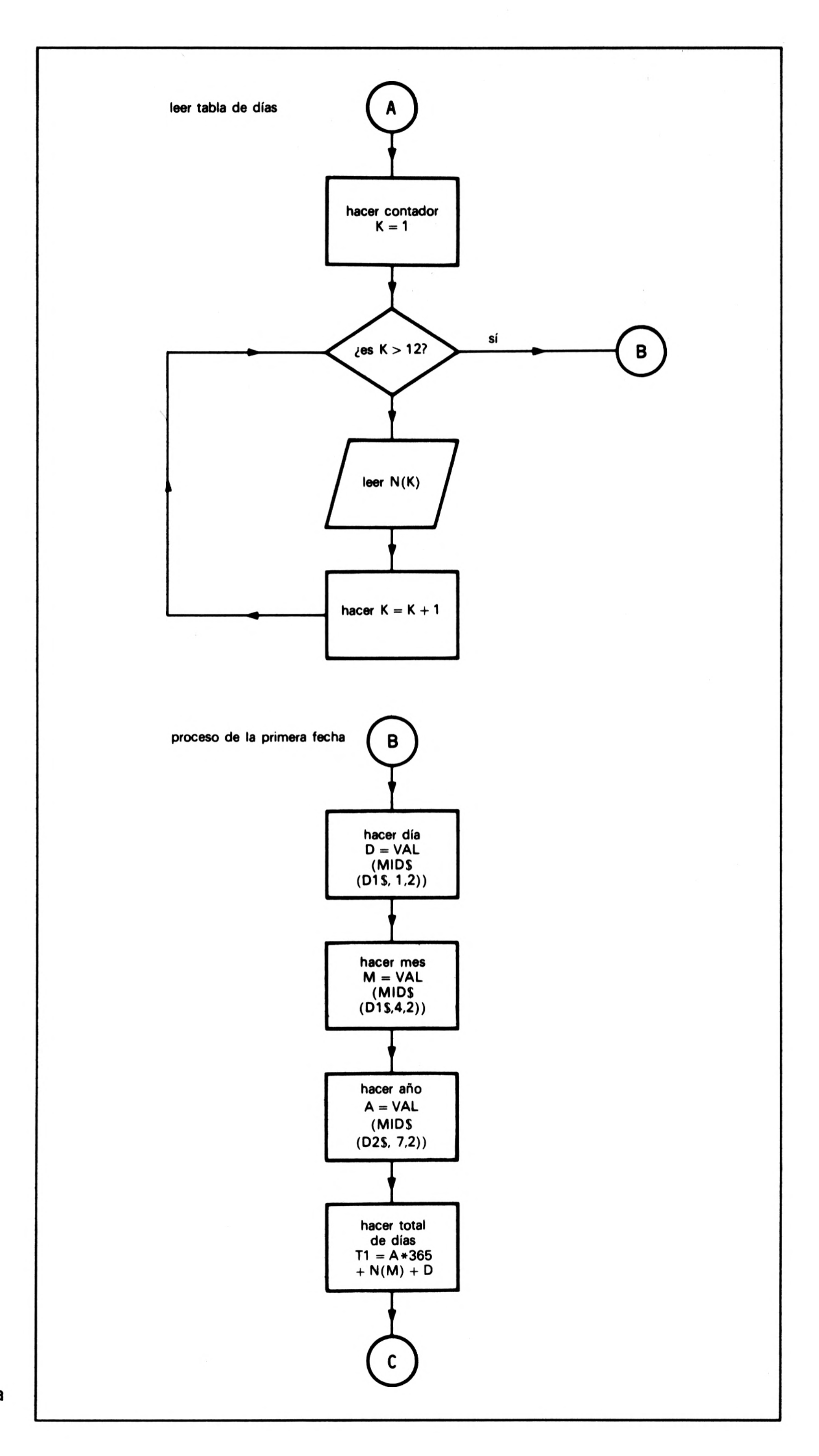
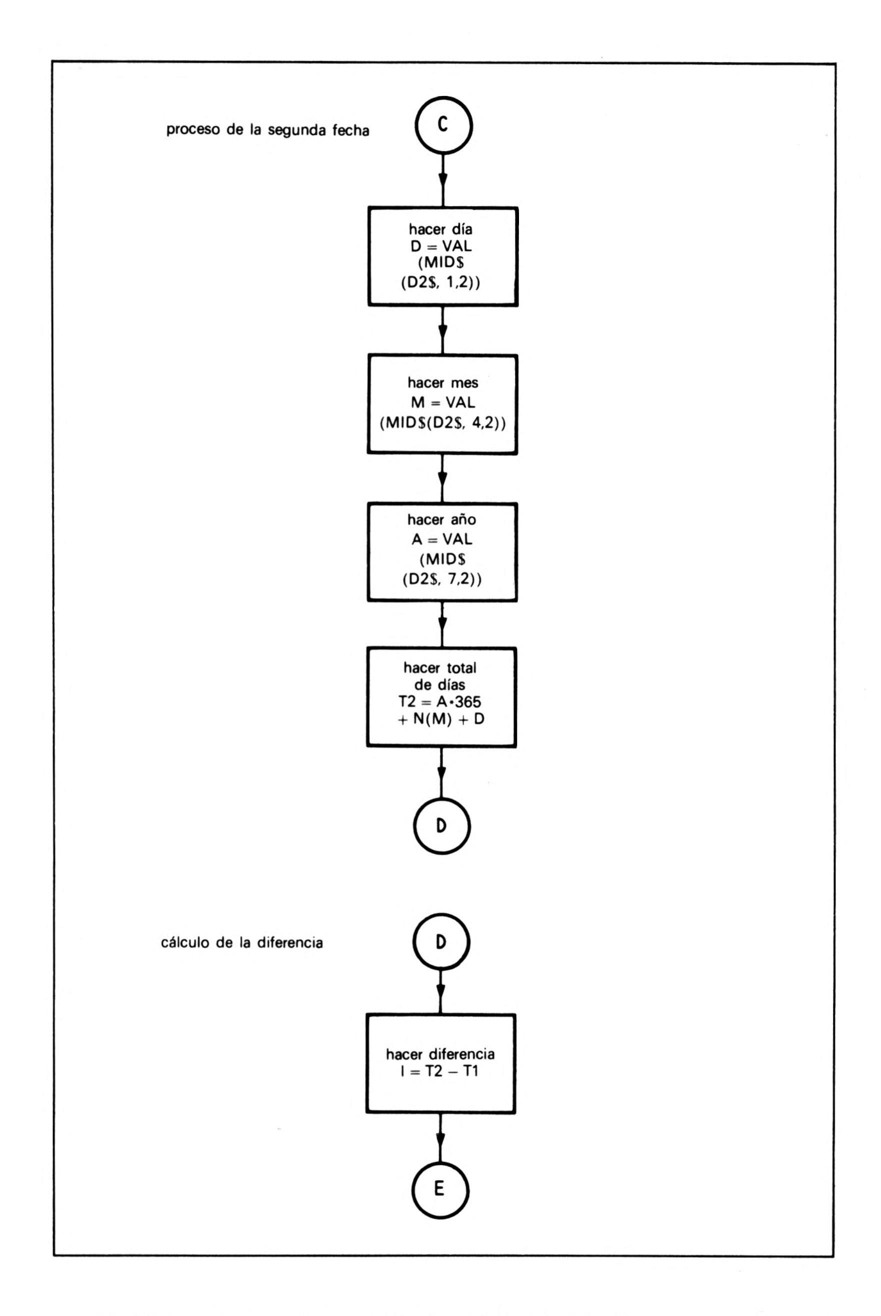


Figura 6.4
Diagrama de flujo del programa ejemplo 6.2



El día se suma a las cantidades obtenidas arriba para obtener el número de días transcurridos desde el principio del siglo.

La tabla que contiene el número de días transcurridos hasta el comienzo de cada mes se archiva como **matriz** de datos (esta estructura de datos se discutirá más detalladamente en el próximo capítulo).

Variables

D1$	Fecha 1 } Para que la diferencia sea positiva,
D2$	Fecha 2 } la fecha 2 es posterior a la 1.
D	Día.
M	Mes.
A	Año.
T1	Días totales desde el cambio de siglo para la fecha 1.
T2	Días totales desde el cambio de siglo para la fecha 2.
I	Intervalo entre las fechas 1 y 2.
K	Contador del bucle.
N(12)	Tabla de días hasta el comienzo de cada mes.

Diagrama de flujo

La figura 6.4 ilustra los pasos del proceso de cálculo de la diferencia entre las dos fechas.

Programa

```
6000 REM PROGRAMA EJEMPLO 6.2
6005 REM CALCULA EL NUMERO DE DIAS COMPRENDIDO
         ENTRE DOS FECHAS
6010 REM
6100 REM LECTURA DE LA TABLA DE DIAS HASTA EL
         COMIENZO DE LOS MESES
6105 DIM N(12)
6110 FOR K = 1 TO 12
6115 READ N(K)
6120 NEXT K
6125 DATA 0, 31, 59, 90, 120, 151
6130 DATA 181, 212, 243, 273, 304, 334
6135 RESTORE
6140 REM
6200 REM SE PROCESA LA PRIMERA FECHA
6205 LET D = VAL(MID$(D1$, 1, 2))
6210 LET M = VAL(MID$(D1$, 4, 2))
6215 LET A = VAL(MID$(D1$, 7, 2))
6220 LET T1 = A*365 + N(M) + D
6225 REM
6300 REM SE PROCESA LA SEGUNDA FECHA
6305 LET D = VAL(MID$(D2$, 1, 2))
6310 LET M = VAL(MID$(D2$, 4, 2))
6315 LET A = VAL(MID$(D2$, 7, 2))
6320 LET T2 = A*365 + N(M) + D
6325 REM
6400 REM CALCULO DE LA DIFERENCIA
6405 LET I = T2 - T1
6410 REM
```

Puntos de interés

- La línea 6105 declara que la variable **N** almacenará una matriz de 12 números.
- El segmento encargado de introducir los números en la matriz **N** debe colocarse al principio del programa, y no en este segmento de manipulación de la fecha. La instrucción **RESTORE** sirve para que este segmento de manipulación pueda utilizarse varias veces.
- En las líneas 6220 y 6320, la variable **N(M)** tiene el valor de "número de días desde el comienzo del año hasta el comienzo del mes M".
- Este programa no está completo, porque carece de segmentos de entrada y salida, y está diseñado para formar parte de otro más amplio. Acepta valores para **D1$** y **D2$** procedentes del resto del programa y entrega el valor de la diferencia en la variable **I** al resto del mismo.
- El programa no incluye ningún procedimiento para verificar si las variables **D1$** y **D2$** son fechas válidas o no; para ello sería preciso añadirle un nuevo segmento.
- Este programa contiene dos partes de estructura idéntica. En el capítulo 9 estudiaremos otra forma de tratar esta situación.

6.13 Conclusión

En este capítulo se han presentado algunos de los recursos de que dispone el Basic para manipular cadenas de caracteres alfanuméricos, junto con varios ejemplos relativos a su aplicación práctica. También se han examinado los conceptos de función, argumento y valor.

- Una función sirve para transformar una cantidad en otra.
- El argumento de una función es la cantidad que ésta transforma.
- Se llama valor de la función al resultado de la transformación operada por la misma.

6 Ejercicio

1. Defina brevemente los términos función, argumento, valor, inversa y cadena nula.
2. Si **C$ = A$ + B$**, ¿qué ocurrirá con **LEN(C$)**, **LEN(A$)** y **LEN(B$)**?

3. Escriba un segmento de un programa en Basic que, cuando reciba un carácter ASCII, entregue el siguiente. Por ejemplo, cuando recibe el carácter **%** (código ASCII 37), debe entregar el **&** (código ASCII 38).
4. Escriba un segmento de un programa en Basic que entregue como resultado los caracteres **ABC ... Z** en una sola línea.
5. Escriba unos módulos de entrada y salida adecuados para poner en funcionamiento el programa ejemplo 6.1. Introduzca una cadena de caracteres y una subcadena, determine la posición de la segunda dentro de la primera y vuelva al principio del programa. Este termina mediante la introducción de una cadena nula.
6. Escriba un segmento de programa capaz de convalidar la fecha en la forma que se usa en el programa ejemplo 6.2. Para ser convalidada, la fecha debe superar las siguientes pruebas:

   La longitud de la cadena alfanumérica correspondiente a la fecha es de 8 caracteres.
   Los caracteres tercero y sexto son /.
   Los restantes caracteres son números.
   El valor de los primeros dos caracteres es inferior a 32.
   El valor de los caracteres cuarto y quinto es inferior a 13.

   El segmento de programa pedido debe aceptar la fecha en forma de variable alfanumérica y devolverla como variable numérica igual a 1 si es válida y a 0 si no lo es.
7. Combine el programa ejemplo 6.2 y el segmento del ejercicio 6 con un módulo de entrada adecuado en un solo programa que acepte la introducción de dos fechas, las convalide y entregue como resultado la diferencia entre ellas. Si cualquiera de las dos fechas resulta ser inválida, será preciso volver a introducirla para que el programa pueda continuar.
8. Modifique el programa ejemplo 6.2 para que tenga en cuenta los años bisiestos. Siga las sugerencias propuestas a continuación o cualquier otro método de su propia cosecha.

   Sugerencias

   Para cualquier año **A**, el número de años bisiestos transcurridos desde el principio del siglo es la parte entera del cociente **A/4** (en Basic, **INT(A/4)**).

   Este es el número de días que deben añadirse al total para tener en cuenta los años bisiestos, salvo que el año en cuestión sea bisiesto y la fecha sea de enero o febrero; en tal caso, ha de sumarse un día menos.

   Verifique el programa calculando el número de días que median entre las fechas

   29/02/84 y 01/03/84
   28/02/83 y 01/03/83
   01/01/84 y 01/01/85
   01/01/85 y 01/01/86

9. Escriba un programa que, dadas una cadena de caracteres y dos subcadenas, sustituya la primera de éstas por la segunda cada vez que aparezca en la serie principal.

   Por ejemplo, dada la cadena "**EN LA LUNA LA RATA MATA A LA GATA**" y las subcadenas "**LA**" y "**UNA**", el programa producirá la cadena "**EN UNA LUNA UNA RATA MATA A UNA GATA**". Utilice el método del programa ejemplo 6.1 para localizar la primera subcadena dentro de la cadena; a continuación, parta la cadena en ese

punto y vuelva a montarla insertando la segunda subcadena en el lugar ocupado por la primera. Repita el proceso cada vez que en la cadena principal aparezca la primera subcadena.

10. Escriba un programa que acepte como entrada una cadena de un texto (contenida en una sola variable) y sustituya todos los conjuntos de dos o más espacios seguidos por espacios únicos.

11. Introduzca el valor de una variable alfanumérica y verifique cada uno de los caracteres de la misma para comprobar si son o no números decimales de una sola cifra. Los códigos ASCII para las diez cifras decimales son los que van del 48 al 57. Si todos los caracteres cumplen la condición de ser cifras decimales, obtenga el valor numérico de la variable alfanumérica; en caso contrario, presente un mensaje que indique que el valor de entrada no es un entero. El programa puede ampliarse para verificar enteros con signo y números con una coma decimal.


RESET
RETURN

7 Matrices

Todo lenguaje de alto nivel ha de ser capaz de manipular datos, pero no sólo a nivel individual, sino también en forma de diversas **estructuras.** El Basic es un tanto pobre en este aspecto, pero de todas maneras dispone de una estructura de datos llamada **matriz.**

En este capítulo examinaremos la naturaleza de la matriz y veremos algunos ejemplos de aplicaciones prácticas de dicha estructura. Nos serviremos para ello de material ya presentado en el texto y además de algunos conceptos y técnicas nuevos que se utilizarán asiduamente en los próximos capítulos. Más en concreto, en los capítulos 13 a 21 se emplean las matrices como bases de ejecución práctica de estructuras de datos más complejas.

7.1 Naturaleza de la matriz

Una **matriz** es un conjunto de datos de la misma naturaleza que se almacenan juntos. Dentro de una matriz cualquiera, el número de datos individuales es fijo. En Basic, la matriz puede ser **numérica** (constituida exclusivamente por números) o **alfanumérica** (constituida

por caracteres alfanuméricos); nunca pueden mezclarse caracteres de los dos tipos mencionados dentro de una misma matriz.

La matriz se crea **declarándola** al principio del programa o del segmento de programa en que vaya a utilizarse. Esta declaración se efectúa por medio de una instrucción **DIM** (abreviatura de DIMENSION). Por ejemplo:

```
DIM A(20), B$(5)
```

declara que la variable **A** es una matriz de 20 números y la **B$** otra de 5 caracteres alfanuméricos.

Cada uno de los datos individuales de una matriz es un **elemento,** que se identifica por un número llamado **índice** o **subíndice** que denota su posición dentro de la matriz.

Por ejemplo: si la matriz **B$** declarada más arriba contiene los elementos

```
EUROPA
ASIA
AMERICA
AFRICA
AUSTRALIA
```

B$(3) es **AMERICA** y **B$(5)** es **AUSTRALIA.**

El índice de la matriz puede ser una variable. Así, **B$(M)** denota el elemento número **M** de **B$**; si **M** = 1, **B$(M)** es **EUROPA.**

Las matrices de índice único se llaman **unidimensionales.** Una matriz puede tener más de una dimensión, aunque en la mayor parte de las versiones del Basic el límite es dos. Más adelante, en este mismo capítulo, hablaremos de estas matrices bidimensionales.

Casi todas las versiones del Basic aceptan el empleo del cero como índice. Por tanto, es correcto referirse a un elemento como **B$(0)** y, estrictamente hablando, una matriz declarada como **X(10)** tiene once elementos. Esta posibilidad no se usa mucho, pero resulta extraordinariamente útil en algunas aplicaciones. En el lenguaje de referencia, se supone que los índices de las matrices parten de cero.

7.2 Algunos usos de las matrices

A continuación se esbozan algunas de las aplicaciones más comunes de esta estructura de datos.

Para lo que más se usan las matrices es para almacenar tablas de referencia. El acceso a los elementos se realiza por medio de los

índices, o bien por escrutinio de la tabla completa hasta localizar el elemento deseado.

También se usan las matrices para archivar grandes colecciones de datos con vistas a realizar operaciones como ordenar o refundir. Estos procesos se discutirán más adelante.

Como las matrices son las únicas estructuras de datos de que dispone el Basic, se emplean como base de otras: pilas, colas, listas y árboles. De estas estructuras se hablará en los capítulos 18 a 21.

Las matrices unidimensionales tienen una propiedad muy útil, y es que su estructura es idéntica a la de una porción de memoria del ordenador; los elementos de la matriz corresponden a los datos individuales de la memoria, y el índice de aquélla a la **dirección** de ésta.

7.3 Programa ejemplo 7.1

La jerga informática está plagada de abreviaturas, como **IBM, ADM, DPM,** etc. El objetivo de este programa es proporcionar un directorio de tales abreviaturas. Cuando se introduce cualquiera de ellas por el teclado, el ordenador da inmediatamente su significado.

Método

Se emplean dos matrices, una para las abreviaturas y otra para los significados. Entre los elementos de ambas matrices se establece una correspondencia tal que el elemento **I** de la segunda almacena el significado de la abreviatura situada en el elemento **I** de la primera. La estructura general del programa consta de tres partes.

En la primera parte se introducen los elementos de las matrices. Estos no tienen por qué estar en orden alfabético. Lo importante es introducir simultáneamente abreviaturas y significados.

En la segunda parte se solicita del usuario que introduzca una abreviatura. Acto seguido se busca la primera matriz hasta localizar un elemento igual a la abreviatura introducida, y se presenta a la salida el significado correspondiente. Si no aparece la abreviatura introducida, se presenta un mensaje.

En la tercera parte se pregunta al usuario si desea introducir alguna otra abreviatura; en caso afirmativo, se repite la segunda parte del programa.

En la segunda parte se recurre a la técnica de bifurcación hacia el exterior de un bucle que ya se expuso en el capítulo 5.

Variables

B$(20) Matriz de abreviaturas.
S$(20) Matriz de significados.
A$ Abreviatura introducida por el usuario.
K, L Contadores de bucles.
C Variable de verificación, que detecta la localización o no de una abreviatura.
I$ Invitación a utilizar de nuevo el programa.

Diagrama de flujo

El flujo de control de este programa se ilustra en la figura 7.1.

Programa

```
1000 REM PROGRAMA EJEMPLO 7.1
1005 REM DICCIONARIO DE ABREVIATURAS
1010 REM
1015 DIM B$(20), M$(20)
1020 PRINT "DICCIONARIO DE ABREVIATURAS"
1025 PRINT
2000 REM PARTE 1: INTRODUCCION DE ABREVIATURAS Y
         SIGNIFICADOS
2005 FOR K=1 TO 20
2010 READ B$(K), M$(K)
2015 NEXT K
2100 REM DATOS PARA ABREVIATURAS, SIGNIFICADO,
         ABREVIATURAS, ETC
2105 DATA "IBM", "''INTERNATIONAL BUSSINES MACHI
          NE '' (FIRMA INTERNACIONAL)"
2110 DATA "ICL", "''INTERNATIONAL COMPUTERS LIMI
          TED'' (FIRMA INTERNACIONAL)"
2115 DATA "ADM", "ACCESO DIRECTO A LA MEMORIA"
2120 DATA "DPM", "ADMINISTRADOR DEL PROCESO DE
          DATOS. ''DATA PROCESSING MANAGER''"
2125 DATA "UCP", "UNIDAD CENTRAL DE PROCESO"
2130 DATA "MV", "MAQUINA VIRTUAL"
2135 DATA "UAL", "UNIDAD ARITMETICO-LOGICA"
2140 DATA "ABD", "ADMINISTRADOR DE UNA BASE DE
          DATOS"
2145 DATA "VDU", "UNIDAD DE REPRESENTACION VISUAL
           .''VISUAL DISPLAY UNIT''"
2150 DATA "EDP", "ELABORACION ELECTRONICA DE
          DATOS. ''ELECTRONIC DATA PROCESSING''"
2155 DATA "HLL", "LENGUAJE DE ALTO NIVEL. ''HIGH
          LEVEL LANGUAGE''"
2160 DATA "SO","SISTEMA OPERATIVO"
```

```
2170 DATA "GIGO", "BASURA DENTRO, BASURA FUERA.
          (JERGA DE ORDENADORES). ''GARBAGE IN,
          GARBAGE OUT''"
2175 DATA "CP/M","PROGRAMA DE CONTROL PARA MICRO
          PROCESADORES"
2180 DATA "SBC","SOCIEDAD BRITANICA DE COMPUTADO
          RAS"
2185 DATA "CNC", "CENTRO NACIONAL DE CALCULO"
2190 DATA "DEC", "''DIGITAL EQUIPMENT CORPORATION
          '' (FIRMA INTERNACIONAL)"
2195 DATA "ASCII", "CODIGO PATRON AMERICANO PARA
          INTERCAMBIO DE INFORMACION. ''AMERICAN
          STANDARD CODE FOR INFORMATION INTER
          CHANGE''"
2200 DATA "CDC", "''CONTROL DATA CORPORATION''
          (FIRMA INTERNACIONAL)"
2205 DATA "JCL", "LENGUAJE PARA EL CONTROL DE TRA
          BAJO. ''JOB CONTROL LANGUAGE''"
2210 REM
3000 REM PARTE 2: ENTRADA Y LOCALIZACION DE ABRE
        VIATURAS, SALIDA DE SIGNIFICADOS
3005 PRINT "ABREVIATURA?";
3010 INPUT A$
3015 LET C = 0
3020 FOR L = 1 TO 20
3025 IF A$ <> B$(L) THEN 3045
```

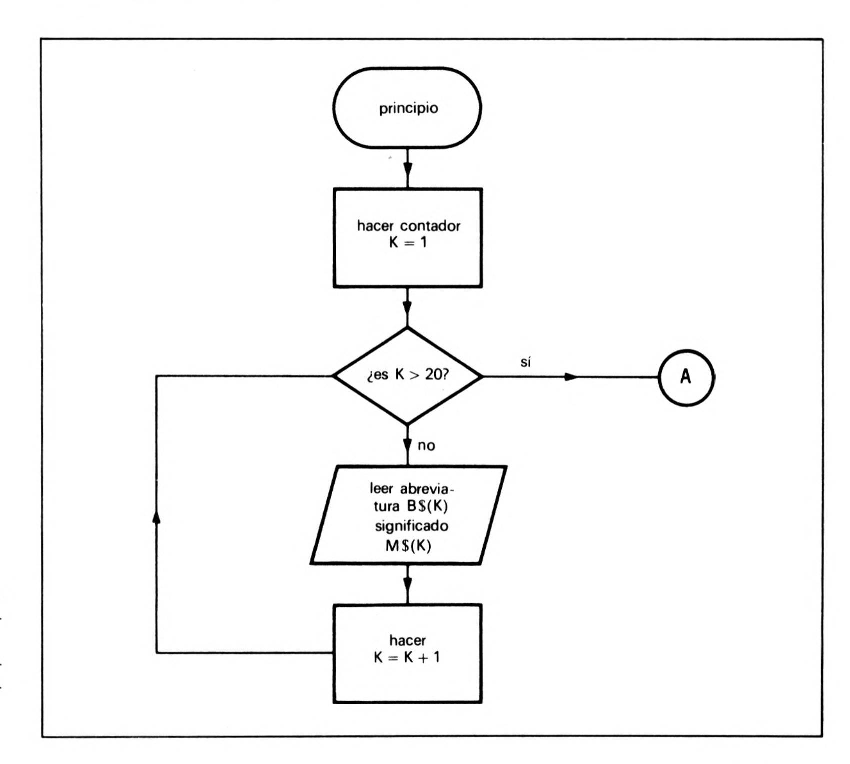
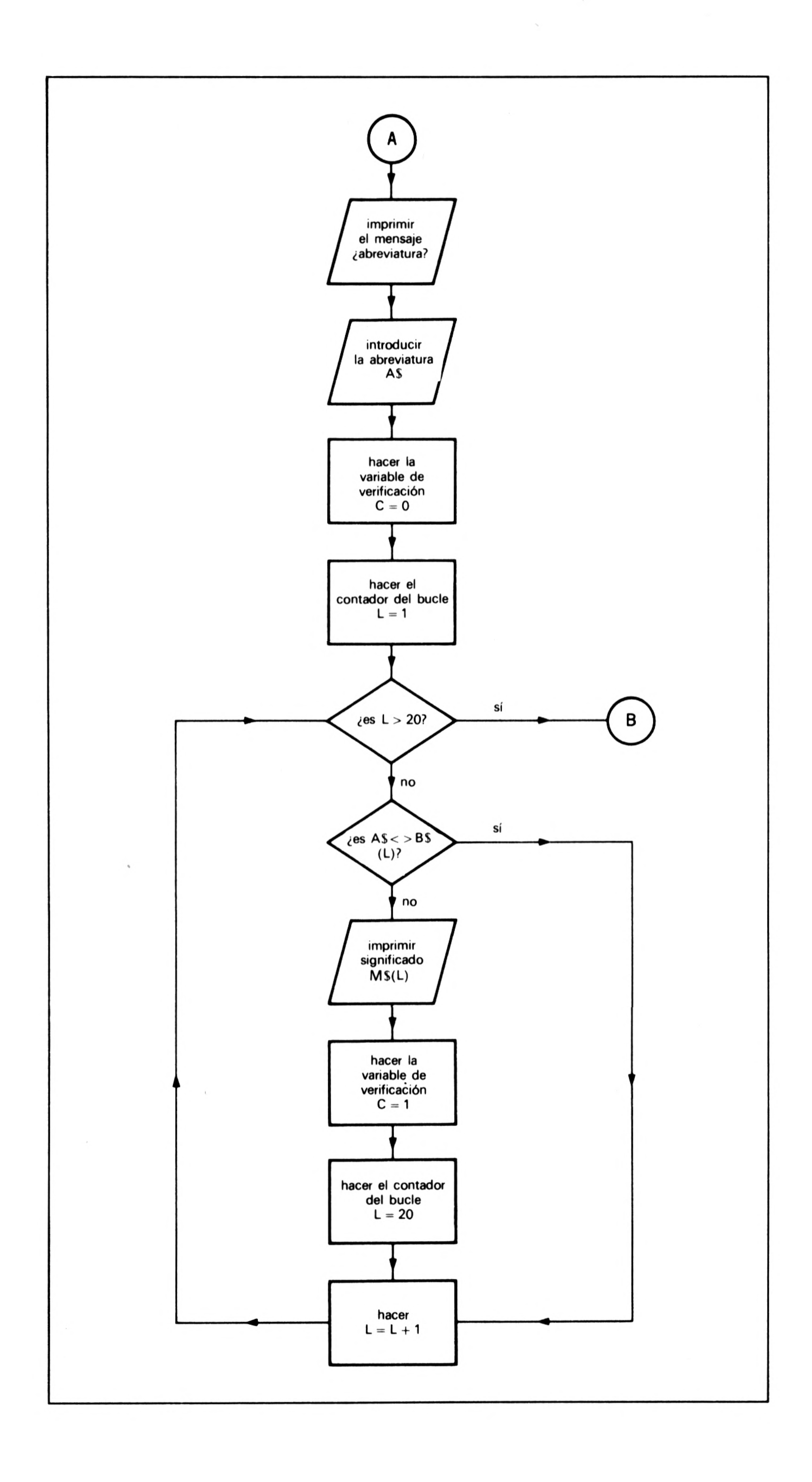
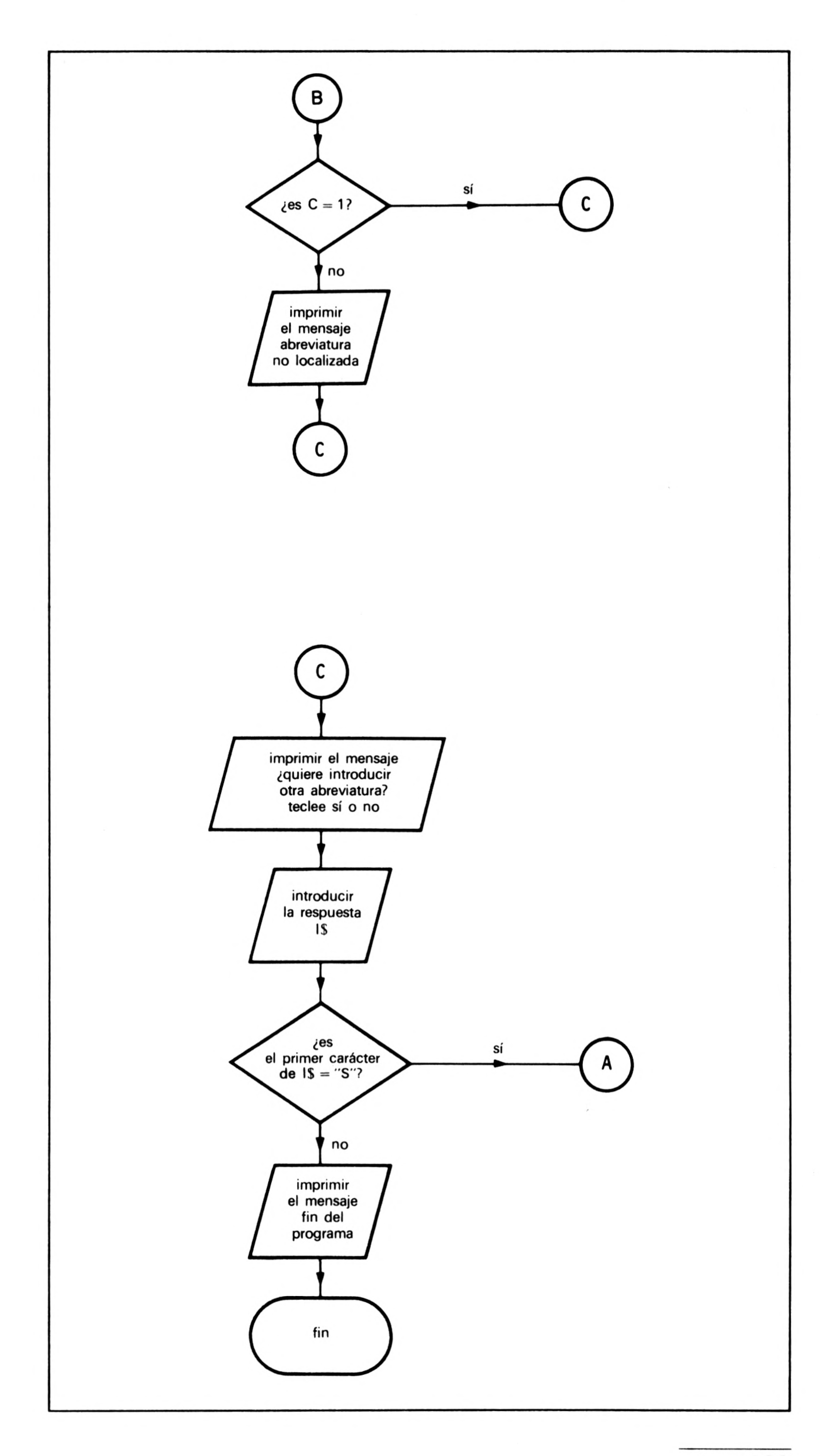


Figura 7.1
Diagrama de flujo del ejemplo 7.1
a) Introducir abreviaturas y significados
b) Introducir y localizar una abreviatura y obtener un significado
c) Decidir si b) se repite o no


A
imprimir
el mensaje
¿abreviatura?
introducir
la abreviatura
A$
hacer la
variable de
verificación
C = 0
hacer el
contador del bucle
L = 1
¿es L > 20?
sí
B
no
¿es A$< >B$
(L)?
sí
no
imprimir
significado
M$(L)
hacer la
variable de
verificación
C = 1
hacer el contador
del bucle
L = 20
hacer
L = L + 1


B
¿es C = 1?
sí
C
no
imprimir
el mensaje
abreviatura
no localizada
C
C
imprimir el mensaje
¿quiere introducir
otra abreviatura?
teclee sí o no
introducir
la respuesta
I$
¿es
el primer carácter
de I$ = "S"?
sí
A
no
imprimir
el mensaje
fin del
programa
fin

```
3030 PRINT M$(L)
3035 LET C= 1
3040 LET L = 20
3045 NEXT L
3050 IF C = 1 THEN 4000
3055 PRINT "ABREVIATURA NO ENCONTRADA"
3060 REM
4000 REM PARTE 3
4005 PRINT
4010 PRINT "DESEA PREGUNTAR OTRA ABREVIATURA?"
4015 PRINT "ESCRIBA SI O NO"
4020 INPUT R$
4025 IF MID$(R$, 1, 1) = "S" THEN 3000
4030 PRINT
4035 PRINT "FIN DEL PROGRAMA"
4040 END
```

Puntos de interés

- Observe detenidamente la correspondencia entre las instrucciones **READ** y **DATA**. Los elementos se leen alternativamente en una y otra matriz.
- La segunda parte del programa busca **B$** elemento por elemento hasta dar con el que se busca. El procedimiento sirve perfectamente para matrices reducidas, pero es muy lento para grandes estructuras. En el capítulo 16 se expondrán otras técnicas de búsqueda.
- En la parte 3, la condición para continuar con el programa es que el primer carácter introducido por el usuario sea **S**.

7.4 Matrices bidimensionales

Una matriz unidimensional puede imaginarse como una columna de elementos con un índice único que identifica por completo la posición de cada uno de ellos dentro de la misma. Esta imagen puede ampliarse a una organización de varias columnas dispuestas de manera que los elementos adyacentes de ellas formen filas. Semejante estructura es lo que se llama una matriz **bidimensional,** que exige dos índices, uno para determinar la fila en que se encuentra el elemento y otro para indicar la columna.

Así, los elementos de la matriz bidimensional **M(3, 4)** podrían disponerse como sigue:

```
M(1,1)  M(1,2)  M(1,3)  M(1,4)
M(2,1)  M(2,2)  M(2,3)  M(2,4)
M(3,1)  M(3,2)  M(3,3)  M(3,4)
```

En general. **M(I, J)** denota el elemento situado en la fila **I** y la columna **J**.

Este concepto está íntimamente relacionado con el matemático de matriz, y la resolución de matrices matemáticas constituye una de las aplicaciones más frecuentes de las matrices bidimensionales, como veremos en el ejemplo que viene a continuación.

7.5
Programa ejemplo 7.2

El programa proporciona los principios de una aplicación general de tratamiento de matrices. Los objetivos de esta aplicación son los siguientes:

La aplicación almacena tres matrices **A, B** y **C** de 10 filas por 10 columnas como máximo, y permite realizar las siguientes operaciones:

Introducir la matriz **A**.
Introducir la matriz **B**.
Sumar las matrices **A** y **B** para obtener la **C**.
Restar **B** de **A** para obtener **C**.
Multiplicar **A** por una constante para obtener **C**.
Multiplicar **A** por **B** para obtener **C**.
Reproducir **C** en **A**.
Reproducir **C** en **B**.
Presentar como salida la matriz **C**.

Por combinación de todas estas operaciones puede llevarse a cabo una serie más amplia de cálculos con las matrices; los resultados de un cálculo pueden utilizarse en los siguientes.

El programa está escrito en forma de varios **módulos,** uno por cada operación. Además, hay un módulo de inicialización y otro de control. Aquí se presentan los módulos de inicialización, de control general, de introducción a la matriz **A**, de suma de **A** y **B** y de salida de **C**. Los restantes quedan como ejercicios; algunos se discuten al final de este mismo capítulo y los otros en los proyectos sugeridos en la última parte del libro.

Método

Se discutirá por separado el método empleado en el diseño de cada uno de los módulos.

Módulo de inicialización

Se declaran las matrices utilizadas en el programa y se presenta al usuario un mensaje de introducción.

Módulo de control

El programa se controla mediante un conjunto de órdenes. Las correspondientes a los módulos descritos aquí son:

INT A Introducir la matriz **A**.
SUM Sumar **A** y **B** para obtener **C**.
SAL Presentar a la salida la matriz **C**.

El módulo de control solicita una orden y transfiere el mando al módulo de operación correspondiente a la orden cargada. Si no reconoce ésta, presenta un mensaje y solicita otra. El control vuelve a este módulo desde el de operación. La orden **FIN** termina el programa.

Introducción de la matriz A

Primero se introducen los números de filas y columnas de la matriz **A**. Si alguno de ellos está fuera del intervalo 1-10 o no es un entero, se solicita la reintroducción del mismo. La matriz se carga por filas, elemento a elemento; es decir, el orden de introducción es **A(1, 1)**, **A(1, 2)**, **A(1, 3)**, etc. Se emplean bucles anidados cuyos contadores actúan como índices. El bucle exterior corresponde a las filas y el interior a las columnas.

Suma de las matrices A y B

Dos matrices pueden sumarse sólo si son del mismo tamaño. Así que, en primer lugar, se comprueba el número de filas y columnas de **A** y **B** y, si no son iguales, se presenta un mensaje y se deja la suma sin efectuar.

La suma de las matrices **A** y **B** es otra matriz **C**, cada uno de cuyos elementos es igual a la suma de los elementos correspondientes de **A** y **B**. Es decir:

```
C(I, J) = A(I, J) + B(I, J)
```

La operación se realiza por filas y columnas de matrices mediante bucles anidados.

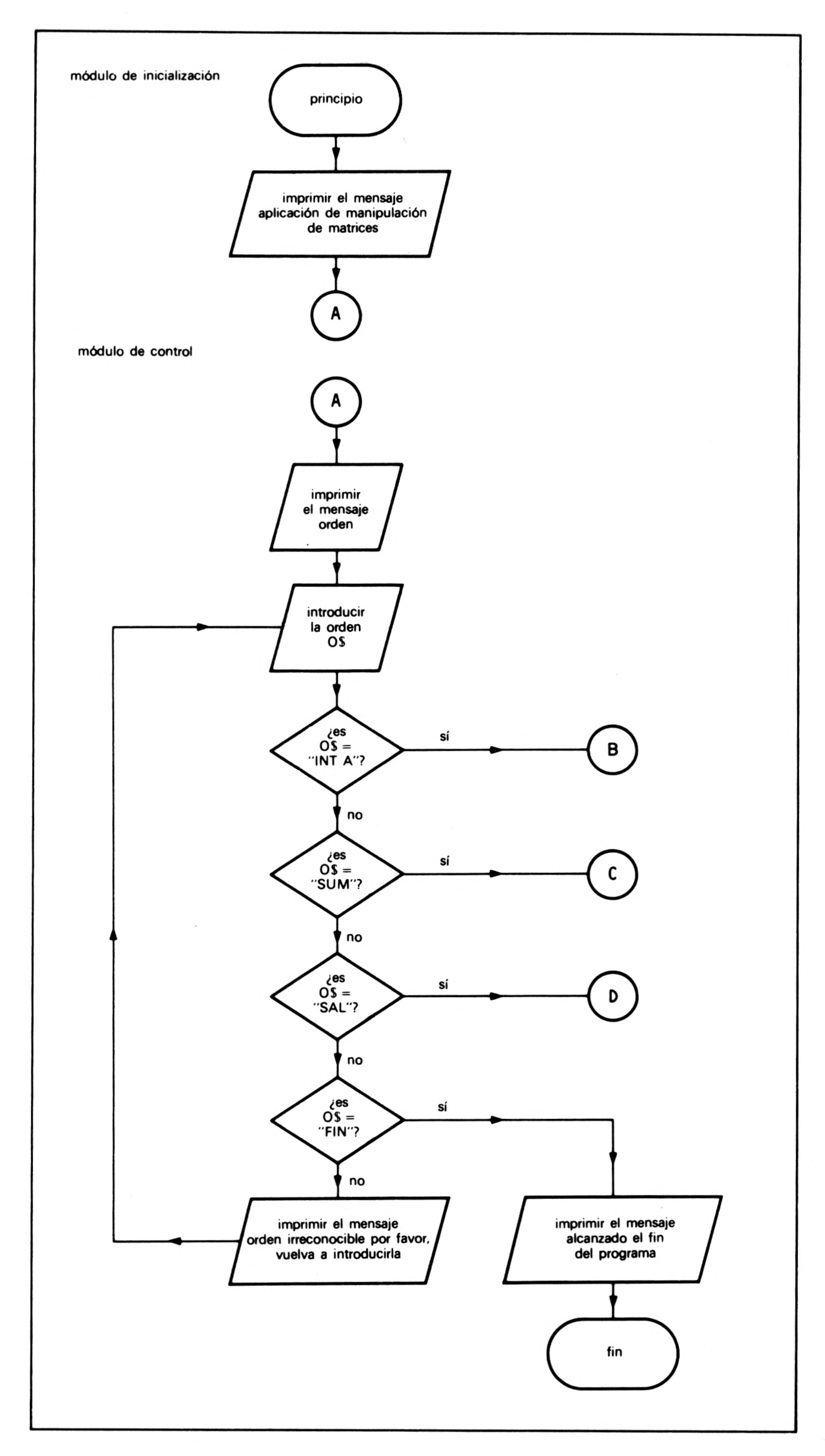
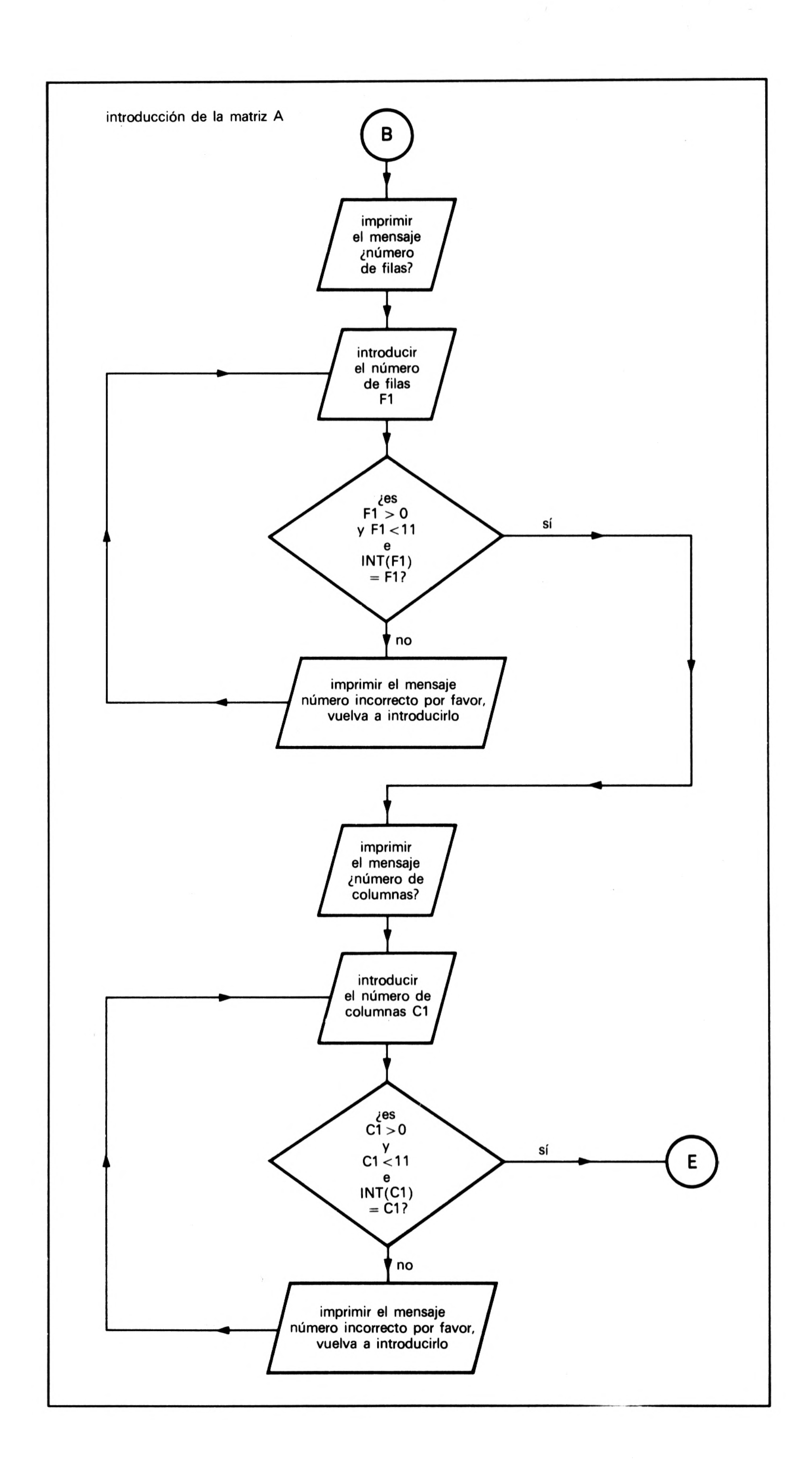
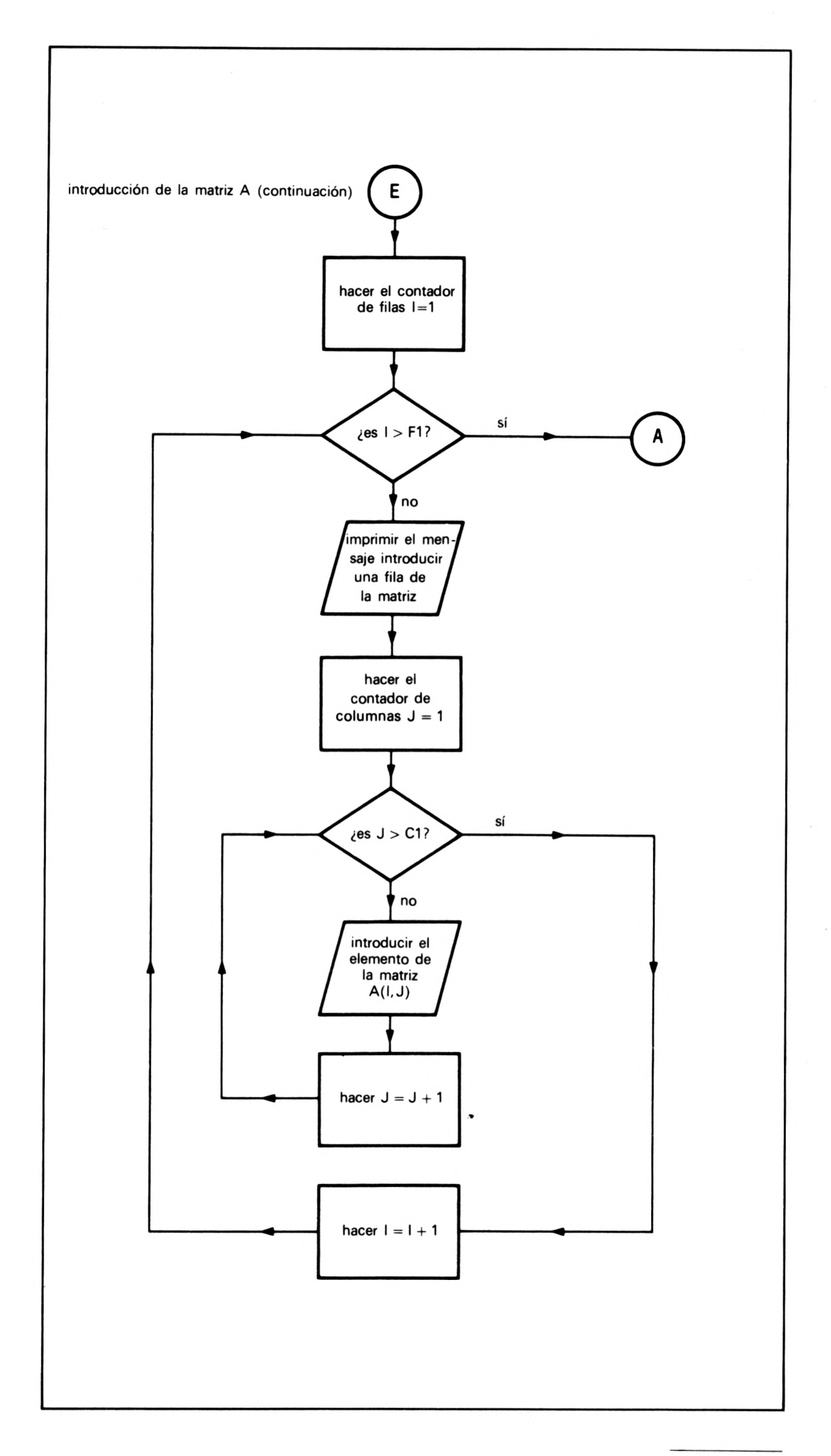
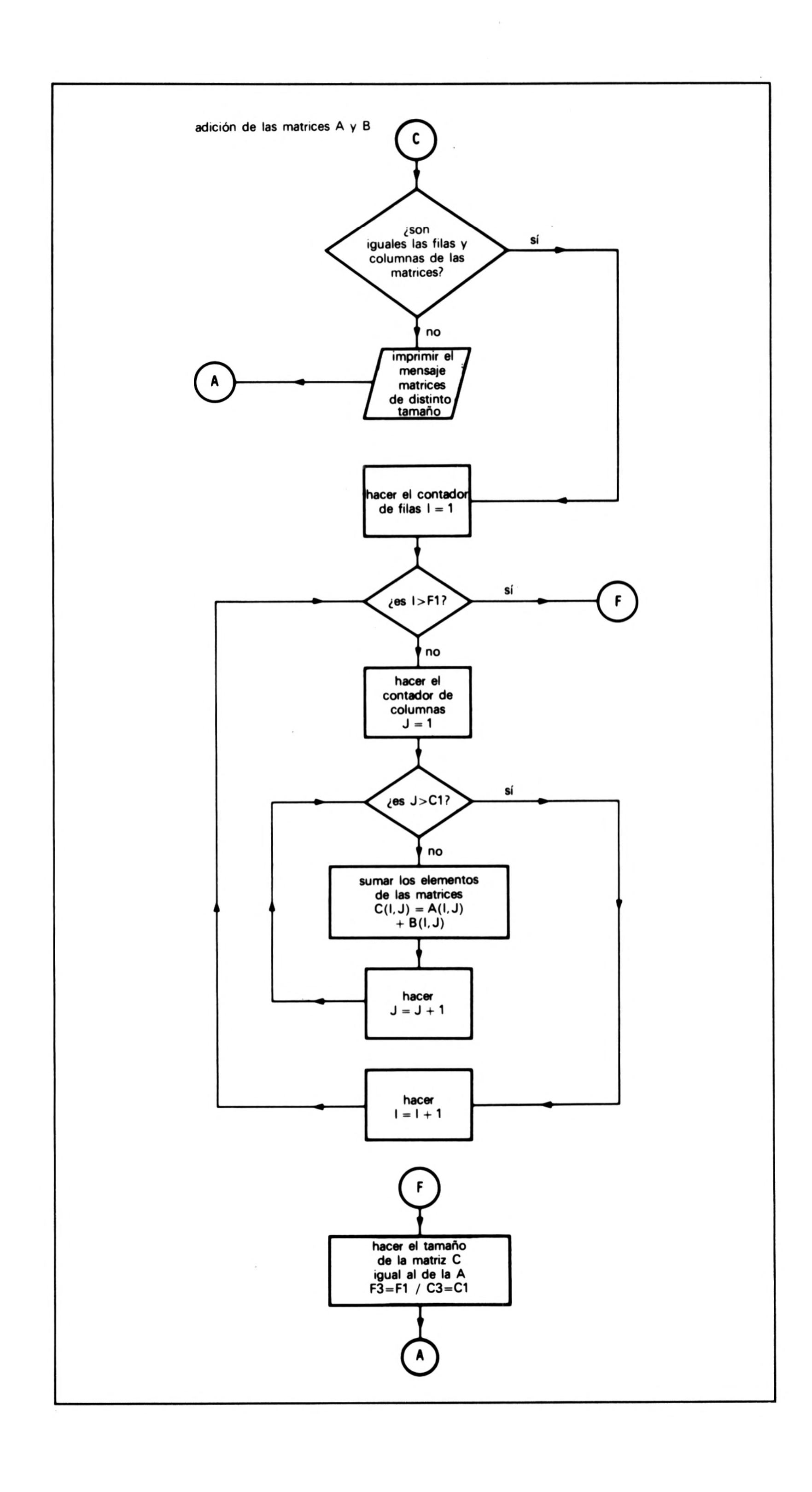
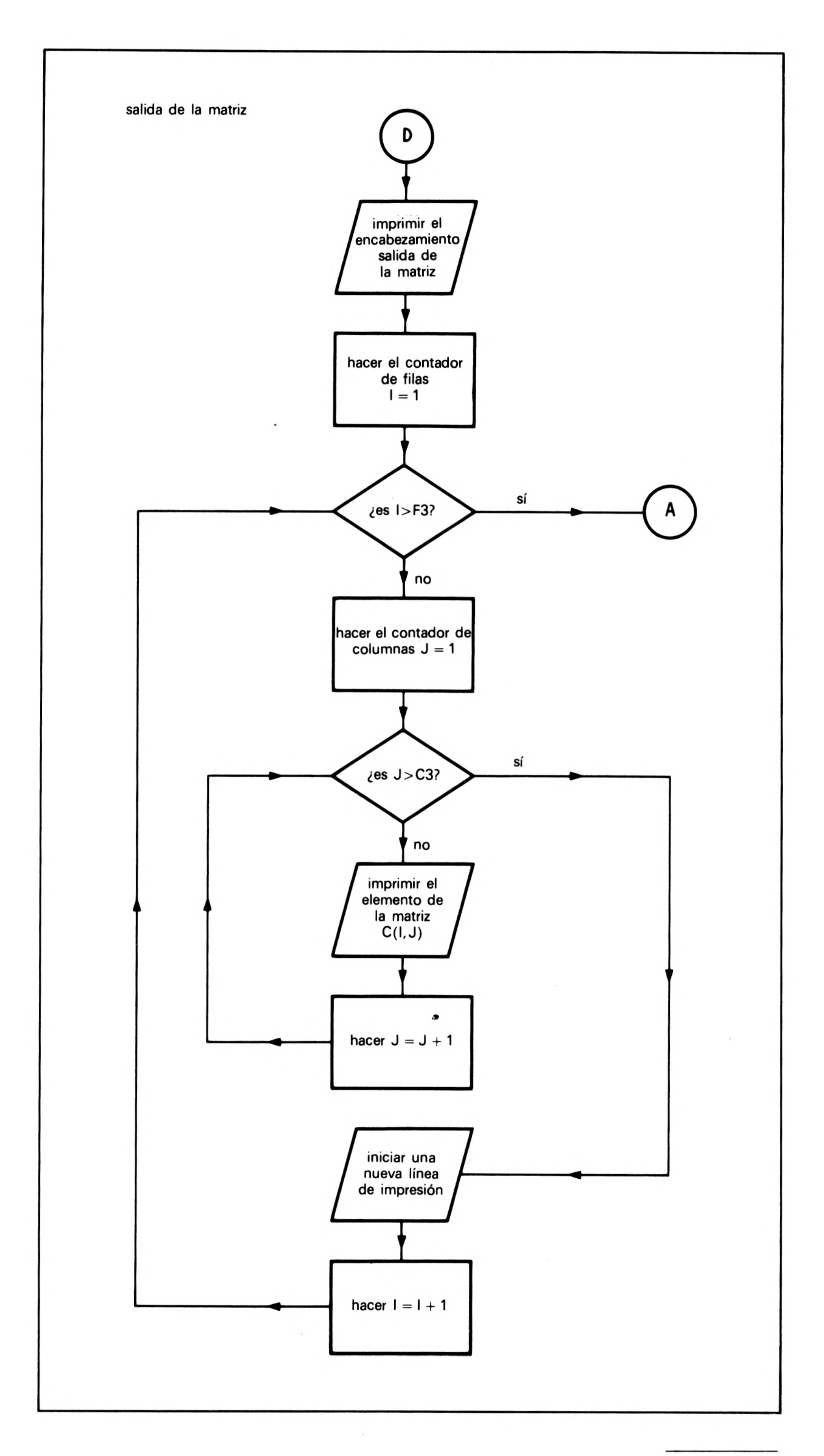


Figura 7.2
Diagrama de flujo del programa ejemplo 7.2


introducción de la matriz A
B
imprimir
el mensaje
¿número
de filas?
introducir
el número
de filas
F1
¿es
F1 > 0
y F1 < 11
e
INT(F1)
= F1?
sí
no
imprimir el mensaje
número incorrecto por favor,
vuelva a introducirlo
imprimir
el mensaje
¿número de
columnas?
introducir
el número de
columnas C1
¿es
C1 > 0
y
C1 < 11
e
INT(C1)
= C1?
sí
E
no
imprimir el mensaje
número incorrecto por favor,
vuelva a introducirlo


introducción de la matriz A (continuación)
E
hacer el contador de filas I=1
¿es I > F1?
sí
A
no
imprimir el mensaje introducir una fila de la matriz
hacer el contador de columnas J = 1
¿es J > C1?
sí
no
introducir el elemento de la matriz A(I, J)
hacer J = J + 1
hacer I = I + 1


adición de las matrices A y B
C
¿son iguales las filas y columnas de las matrices?
sí
no
imprimir el mensaje matrices de distinto tamaño
A
hacer el contador de filas I = 1
¿es I>F1?
sí
F
no
hacer el contador de columnas J = 1
¿es J>C1?
sí
no
sumar los elementos de las matrices C(I, J) = A(I, J) + B(I, J)
hacer J = J + 1
hacer I = I + 1
F
hacer el tamaño de la matriz C igual al de la A F3=F1 / C3=C1
A


salida de la matriz
D
imprimir el encabezamiento salida de la matriz
hacer el contador de filas I = 1
¿es I>F3?
sí
A
no
hacer el contador de columnas J = 1
¿es J>C3?
sí
no
imprimir el elemento de la matriz C(I,J)
hacer J = J + 1
iniciar una nueva línea de impresión
hacer I = I + 1

Salida de la matriz C

La matriz **C** aparece en la salida fila por fila. También en esta ocasión se usan bucles anidados.

Variables

A(10, 10), B(10, 10), C(10, 10) Matrices.
O$ Orden.
I, J Contadores de bucle.
F1, C1 Número de filas y columnas de la matriz **A**.
F2, C2 Número de filas y columnas de la matriz **B**.
F3, C3 Número de filas y columnas de la matriz **C**.

Diagrama de flujo

La figura 7.2 ilustra el flujo de control de los tres módulos del programa.

Programa

```
1000 REM PROGRAMA EJEMPLO 7.2
1005 REM PAQUETE DE APLICACION DE MANIPULACION DE
         MATRICES
1010 REM
1500 REM MODULO DE INICIALIZACION
1505 DIM A(10, 10), B(10, 10), C(10, 10)
1510 PRINT "PAQUETE DE MANIPULACION MATRICIAL"
1520 REM
2000 REM MODULO DE CONTROL
2005 PRINT "ORDEN?";
2010 INPUT O$
2015 IF O$ = "INT A" THEN 2500
2025 IF O$ = "SUM" THEN 3500
2060 IF O$ = "SAL" THEN 7000
2065 IF O$ = "FIN" THEN 2080
2070 PRINT "ORDEN NO RECONOCIDA. POR FAVOR
            REPITA";
2075 GOTO 2010
2080 PRINT "SE HA LLEGADO AL FINAL DEL PROGRAMA"
2085 STOP
2500 REM ENTRADA DE LA MATRIZ A
2505 PRINT "NUMERO DE FILAS?";
2510 INPUT R1
```

```
2515 IF R1 > O AND R1 < 11 AND INT(R1) = R1 THEN
     2530
2520 PRINT "NUMERO INCORRECTO. POR FAVOR REINTRO
              DUZCALO";
2525 GOTO 2510
2530 PRINT "NUMERO DE COLUMNAS?";
2535 INPUT C1
2540 IF C1 > O AND C1 < 11 AND INT(C1) = C1 THEN
     2555
2545 PRINT "NUMERO INCORRECTO. POR FAVOR VUELVA A
              INTRODUCIRLO"
2550 GOTO 2535
2555 FOR I = 1 TO R1
2560 PRINT "INTRODUZCA UNA FILA DE LA MATRIZ"
2565 FOR J = 1 TO C1
2570 INPUT A(I, J)
2575 NEXT J
2580 NEXT I
2585 GOTO 2000
2590 REM
3500 REM SUMA DE LAS MATRICES A Y B
3505 IF R1 = R2 AND C1 = C2 THEN 3520
3510 PRINT "LAS MATRICES NO SON DEL MISMO ORDEN"
3515 GOTO 2000
3520 FOR I = 1 TO R1
3525 FOR J = 1 TO C1
3530 LET C(I, J) = A(I, J) + B(I, J)
3535 NEXT J
3540 NEXT I
3545 LET R3 = R1
3550 LET C3 = C1
3555 GOTO 2000
3560 REM
7000 REM SALIDA DE LA MATRIZ C
7005 PRINT "SALIDA DE LA MATRIZ"
7010 PRINT
7015 FOR I = 1 TO R3
7020 FOR J = 1 TO C3
7025 PRINT C(I, J); " ";
7030 NEXT J
7035 PRINT
7040 NEXT I
7045 GOTO 2000
9000 END
```

Puntos de interés

- Observe atentamente la estructura general del programa, sobre todo la forma en que el módulo de control se relaciona con los otros.
- En el módulo de control se han dejado espacios amplios entre

los números de las líneas para insertar posibles instrucciones adicionales.

- También se han previsto espacios para módulos adicionales.
- Los números de línea reflejan la estructura general del programa.
- La disposición de la salida es mejorable.

7.6 Conclusión

En este capítulo hemos analizado la única estructura de datos de que dispone el Basic: la matriz. Se han presentado técnicas de introducción, proceso y salida de matrices uni y bidimensionales.

Las matrices se utilizarán frecuentemente a lo largo del texto y, más en concreto, constituyen la base de la representación de otras estructuras de datos, como pilas, colas, listas y árboles.

Los puntos más destacables de este capítulo son:

- Una matriz es un conjunto de datos de la misma naturaleza almacenados juntos.
- Dentro de una matriz, cada uno de los elementos se localiza por medio de uno o más índices.
- Se llama dimensión de una matriz al número de índices necesarios para localizar un elemento.
- Una matriz bidimensional tiene la estructura de una matriz matemática.

7 Ejercicio

1. Defina brevemente los siguientes términos: matriz, declaración, elemento, índice.
2. Explique cómo se accede a un elemento de una matriz.
3. Modifique el programa ejemplo 7.1 para convertirlo en otro que traduzca palabras de un idioma a otro y viceversa.
4. Redacte el módulo del programa ejemplo 7.2 encargado de introducir los elementos de la matriz **B**. Corrija el módulo de control para incorporar este nuevo módulo.
5. Las matrices unidimensionales se usan entre otras cosas para almacenar **vectores.** Un vector es un conjunto de números que frecuentemente se interpreta como las coordenadas de un punto en el espacio. Así, el vector **P**, de elementos 5, 7 y 8, representa un punto cuyas coordenadas espaciales son $x = 5$, $y = 7$ y $z = 8$.

Diseñe una aplicación de manipulación de vectores de tres elementos, con módulos para realizar las siguientes operaciones:

Introducir el vector **P**.
Introducir el vector **Q**.
Sumar los vectores **P** y **Q** para obtener el **R**.
Restar el vector **Q** al vector **P** para obtener el **R**.
Multiplicar el vector **P** por una constante para obtener el **R**.
Reproducir el vector **R** en el **P**.
Reproducir el vector **R** en el **Q**.
Presentar en la salida el vector **R**.

6. La multiplicación de dos matrices sólo puede efectuarse cuando el número de columnas de la primera es igual al de filas de la segunda. Para dos matrices **A** y **B**, esta condición supone que si la dimensión de **A** es **A(K, L)**, la de **B** ha de ser **B(L, M)**.

Teniendo en cuenta esta circunstancia, si **C** es el producto de **A** por **B**, cada uno de sus elementos vendrá determinado por la ecuación

```
C(I, J) = A(I, 1) * B(1, J) + A(I, 2) * B(2, J)
                   + ... + A(I, L) * B(L, J)
```

La matriz **C** tiene, pues, las dimensiones **C(K, M)**. Por ejemplo: si las dimensiones de **A** son **A(4, 5)** y las de **B** son **B(5, 3)**, las del producto serán **C(4, 3)**.

Un módulo de programa de multiplicación de matrices precisa de tres bucles anidados, de los que los dos exteriores sirven para las filas y columnas de la matriz producto. Son similares a los bucles anidados de entrada, suma y salida de matrices. El bucle más interno de los tres acumula la suma definida en la ecuación anterior. Fuera de este bucle, el elemento **C(I, J)** se pone a cero, y dentro del mismo se suma al total el producto de cada par de elementos de **A** y **B**.

Utilice estas sugerencias o desarrolle una técnica propia para escribir el módulo del programa ejemplo 7.2 encargado de multiplicar matrices. Antes de empezar el proceso de multiplicación, verifique si los términos del producto son compatibles, es decir, si el número de columnas de la primera matriz es igual al de filas de la segunda. Corrija el módulo de control para incorporar el de multiplicación.

7. *a*) A veces es conveniente almacenar en una matriz una cadena de caracteres que se van a presentar en una línea de la pantalla. Declare una matriz que tenga el mismo número de caracteres que una línea de la pantalla de su ordenador. Escriba un módulo de programa que presente los caracteres de la matriz en la pantalla.

b) Una ampliación de esta idea consiste en utilizar una matriz bidimensional para almacenar todos los caracteres de una pantalla. Las filas y columnas de la matriz corresponderán a las filas y columnas de la pantalla. Declare una matriz bidimensional del tamaño adecuado y escriba módulos de programa para introducir o generar los caracteres de la matriz y presentarlos como salida en la pantalla.

8. Utilice el método del programa ejemplo 7.1 para crear un listín telefónico personal. Archive en matrices los nombres de una serie de personas y sus números de teléfono. Al introducir un nombre debe aparecer en la salida su número de teléfono.

9. Almacene en una matriz unidimensional una serie de nombres de ciudades, y en otra bidimensional las distancias entre ellas. Asegúrese de

que en ambas el orden de las ciudades es el mismo. Al introducir los nombres de dos ciudades se obtiene, si ambas están en la matriz, la distancia que las separa.

10. Utilice una matriz bidimensional para archivar los resultados de una liga futbolística. Escriba los módulos de programa necesarios para cargar las cifras iniciales en la matriz, actualizarlas y presentar en pantalla la tabla completa o parte de ella.

11. Introduzca una serie de números en una matriz y obtenga a la salida el mayor (o el menor, o ambos) de ellos.


RESET
RETURN

8 Funciones

Este capítulo reúne algunas de las posibilidades que ofrece el Basic en materia de **funciones.** Se explica también la naturaleza de una función y se presentan las funciones contenidas en el lenguaje de referencia. El capítulo termina con una discusión en torno a las funciones definidas por el usuario.

Ya en el capítulo 6 se hizo referencia al concepto de éstas con motivo del análisis de algunas funciones de manipulación de caracteres. Ahora investigaremos este tópico desde un punto de vista más general.

Los programas de este capítulo utilizan elementos del Basic presentados ya en otros anteriores; a su vez, el material nuevo que aporten estas páginas se empleará con regularidad hasta el final del texto.

8.1 Naturaleza de una función

En su forma más general, una función puede considerarse como una relación entre dos cantidades o como una transformación de una

cantidad en otra. Muchas funciones tratan con números, pero no son las únicas que existen.

Por ejemplo, considere una función que asocie una letra al entero que indica su posición en el alfabeto; esta función asociaría la letra A al número 1, la B al 2, etc. Este mismo ejemplo se usará para identificar varias características de las funciones que discutiremos a continuación.

En primer lugar, una función tiene un **nombre.** Un nombre apropiado para la que nos sirve de ejemplo sería **POSN** (posición **N**). La cantidad sobre la que opera la función se llama **argumento** de la misma. En el caso de la función **POSN**, el argumento puede ser cualquier letra del alfabeto. Por lo general se escribe entre paréntesis tras el nombre de la función; así, el argumento de **POSN(X)** es **X**.

La mayor parte de las funciones se definen únicamente para un conjunto específico de argumentos. En el caso de la función **POSN**, ese conjunto es el de las letras del alfabeto. Si se adjunta a una función un argumento situado fuera de ese conjunto definido, se produce una situación de error; así, la función **POSN("?")** no puede definirse. En Basic, y en cualquier otro lenguaje de programación, hay que tener muy en cuenta este último punto.

Una función asocia su argumento a otra cantidad que se llama **valor** de la función. Por ejemplo, el valor de **POSN("X")** es 24. Cuando en un programa se utiliza una función, se establecen unidos su nombre y su argumento. Este puede ser una constante o una variable. Si es una variable, ha de tomar un valor específico en el punto del programa en que se encuentra la función.

El ordenador, al ejecutar el programa, **evalúa** la función. En otras palabras: asocia el argumento al valor que le corresponde, valor que pasa a ocupar el lugar del nombre de la función en ese punto del programa. Por ejemplo, si **POSN** fuese una función usada en Basic, la instrucción

```
LET A = POSN("X")
```

daría lugar a la evaluación de la misma y a la asignación de su valor (24) a la letra variable **A**. El argumento **X** aparece entre comillas para señalar claramente que se trata de una constante alfanumérica y no de una variable numérica.

En resumen: una función tiene un nombre y asocia una cantidad —su argumento— a otra —su valor—. Este proceso de asociación se denomina evaluación de la función. En su forma más general, una función puede tener más de un argumento, pero nunca más de un valor por cada argumento o conjunto de argumentos.

ABS(X)	Valor absoluto de X.
EXP(X)	Función exponencial e^x, siendo $e = 2{,}71828$.
LOG(X)	Logaritmo natural de X, es decir, en base *e*, siendo $e = 2{,}71828$.
INT(X)	El mayor entero no superior a X.
RND(X)	Un número al azar entre 0 y 1 que depende indirectamente del valor de X.
SGN(X)	Vale + 1 si X es positivo 0 si X es cero − 1 si X es negativo.
SQR(X)	Raíz cuadrada de X.
SIN(X)	Seno del ángulo X expresado en radianes.
COS(X)	Coseno del ángulo X expresado en radianes.
TAN(X)	Tangente del ángulo X expresado en radianes.
ATN(X)	Angulo en radianes cuya tangente es X.
TAB(X)	Función de tabulación; imprime el siguiente carácter X posiciones a partir del comienzo de la línea.

Nota

El comportamiento de la función de números aleatorios RND depende del tipo de ordenador que la ejecute.

Figura 8.1
Funciones normalizadas del lenguaje de referencia

8.2 Funciones normalizadas del Basic

La figura 8.1 recoge las funciones disponibles en la versión de referencia del Basic. Estas son las **funciones normalizadas,** llamadas así para diferenciarlas de las definidas por el usuario, que se discutirán más adelante.

A éstas habría que añadir las de manipulación de caracteres, que ya se estudiaron en el capítulo 6. Como aquélla, esta lista se ha resumido para que sólo incluya las funciones disponibles en la mayor parte de las versiones del Basic. Las próximas secciones explicarán brevemente la naturaleza de las funciones recogidas en la figura.

8.3 ABS(X)

La función **ABS** determina el valor absoluto de su argumento. Por ejemplo, **ABS(5)** = 5 y **ABS(−7)** = 7. En cuanto al intervalo de números compatibles con esta función, no hay más limitaciones que las del tamaño del número más grande que es capaz de manejar cada ordenador concreto.

8.4

EXP(X)

La función exponencial **EXP** eleva el número *e* a la potencia indicada por su argumento. El valor de *e* con cinco decimales es 2,71828. Por ejemplo:

```
EXP(1)    = 2,71828
EXP(0)    = 1
EXP(4)    = 54,591
EXP(-1)   = 0,36789
```

Para argumentos positivos, el valor de **EXP(X)** aumenta muy rápidamente con **X**, por lo que el argumento tiene un límite que depende del ordenador con que se trabaja.

Para argumentos negativos, el valor de **EXP(X)** tiende a cero conforme disminuye el valor de **X** (es decir, conforme tiende a menos infinito). Por tanto, en este caso, el límite del argumento **X** será el límite general impuesto por el número de mayor tamaño que sea capaz de soportar la versión concreta del Basic con que se opera.

La función exponencial resulta muy útil cuando se trabaja con cantidades que aumentan o disminuyen a un ritmo regular, como las referidas al crecimiento de la población o a la degradación radiactiva.

8.5

LOG(X)

La función **LOG(X)** genera el logaritmo natural de su argumento, es decir, el algoritmo de base *e*, siendo $e = 2{,}71828$. Por ejemplo:

```
LOG(2.71828)   = 1
LOG(1)         - 0
LOG(54.591)    = 4
LOG(0.36789)   = -1
```

La definición de logaritmo es la siguiente:

> Se llama logaritmo de un número **X** a la potencia a que es necesario elevar la base (en este caso *e*) para obtener dicho número **X**.

De esta definición y de los ejemplos propuestos se deduce que la

función **LOG** es **inversa** de la **EXP**. En otras palabras: la función **LOG** opera en dirección contraria a la **EXP**, lo que significa que

```
LOG(EXP(X)) = X
```

El argumento de la función **LOG** sólo puede ser positivo. No se admiten el cero ni los números negativos.

8.6
INT(X)

La función de redondeo **INT** produce el mayor entero de valor no superior a su argumento. Por ejemplo:

```
INT(2.5)        = 2
INT(3.99999)    = 3
INT(10)         = 10
INT(-2.1)       = -3
```

El argumento de esta función **INT** no está sometido a limitaciones especiales.

La función **INT** es muy útil para determinar si el valor de una variable es o no entero. En efecto, basta la condición

```
IF INT(X) = X
```

para hacer la comprobación. Otra aplicación frecuente de la función **INT** es redondear una cifra a su valor entero más próximo. La instrucción

```
LET Y = INT(X + 0.5)
```

convierte el número **X** en el entero **Y** más próximo.

8.7
Programa ejemplo 8.1

Una sustancia radiactiva emite radiaciones con una intensidad que disminuye con el tiempo según la fórmula

$$r = r_0 e^{-(\log_e(2)t/T)}$$

donde t es el tiempo, r_0 la intensidad original de emisión (a $t = 0$), r es la intensidad actual de emisión y T es la **vida media** de la sustancia o tiempo que tarda la intensidad de emisión en reducirse a la mitad de su valor inicial.

El objetivo de este programa es calcular, a partir de la vida media de una sustancia determinada, la intensidad de emisión de la misma a intervalos regulares hasta que haya caído por debajo del 1 por 100 de su valor inicial.

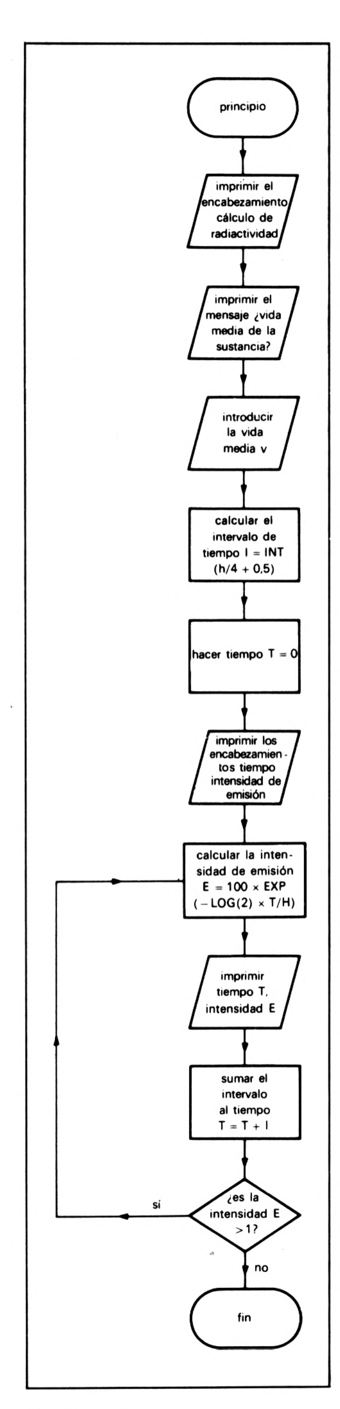


Figura 8.2
Diagrama de flujo del programa ejemplo 8.1

Método

Las especificaciones de este programa son un tanto vagas, caso que es frecuente en la práctica. Por ello, es preciso tomar algunas decisiones previas antes de fijar la metodología.

En primer lugar, hay que determinar un valor para esos "intervalos regulares". Teniendo en cuenta que han de transcurrir varias vidas medias antes de que la actividad descienda por debajo del 1 por 100 de la inicial, puede concluirse que un cuarto de la vida media —redondeado a su valor entero más próximo— constituye un período de tiempo adecuado.

La instrucción

```
LET I = INT(V/4 + 0.5)
```

donde **I** es el intervalo de tiempo y **V** la vida media, se encargará de fijar el intervalo deseado.

La condición "el 1 por 100 de su valor inicial" resulta más manejable si la tasa inicial de emisión se sitúa en 100 y se determina el momento en que se hace inferior a 1.

La estructura general del programa es un bucle que se "repite hasta que", ya estudiado en la sección 5.11. El algoritmo para realizar el programa es el siguiente:

Introducir la vida media.
Calcular el intervalo de tiempo "adecuado".
Poner el tiempo actual a cero.
Repetir el proceso:

Calcular la intensidad de emisión actual.
Indicar a la salida el tiempo y la intensidad de emisión actuales.
Sumar el intervalo de tiempo al tiempo actual.

Hasta que la intensidad de emisión sea inferior a 1.

Una versión en Basic de la fórmula para calcular la intensidad de emisión actual sería:

```
LET E = 100*EXP(-LOG(2)*T/V)
```

donde **E** es la intensidad de emisión actual, **T** es el tiempo y **V** la vida media. La intensidad de emisión inicial es 100.

Diagrama de flujo

La figura 8.2 ilustra el flujo de control de este programa.

Programa

```
100 REM PROGRAMA EJEMPLO 8.1
105 REM CALCULOS DE RADIACTIVIDAD
110 REM
200 PRINT "CALCULOS DE RADIACTIVIDAD"
205 PRINT
210 INPUT "VIDA MEDIA?", V
220 LET I = INT (H/4 + 0.5)
225 LET T = 0
230 PRINT "TIEMPO", "INTENSIDAD DE EMISION"
240 REM
300 LET E = 100*EXP(-LOG(2)*T/V)
305 PRINT T, E
310 LET T = T + I
315 IF E > 1 THEN 300
320 END
```

Puntos de interés

- La condición de la línea 315 es de continuación del bucle. En esta instrucción se ha "dado la vuelta" a la condición "repetir hasta que".
- El bucle "repetir hasta que" va desde la línea 300 hasta la 315.
- Desde cierto punto de vista, este programa es muy ineficaz. Al final del capítulo hay un ejercicio que analiza el origen de esa ineficacia.

8.8 RND(X)

La función generadora de números al azar, **RND**, produce un número al azar comprendido entre 0 y 1. El número se extrae de una **distribución uniforme,** lo que significa que, en teoría, todos los valores del intervalo tienen las mismas probabilidades de salir.

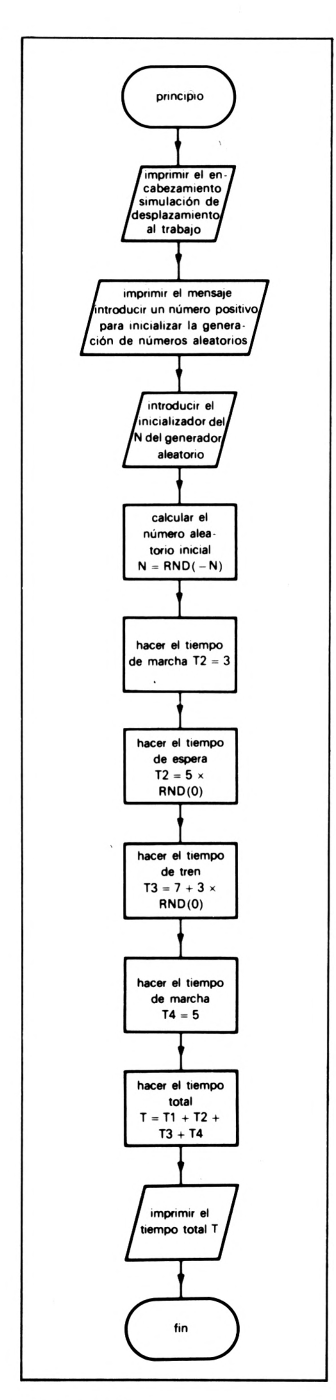


Figura 8.3
Diagrama de flujo del programa ejemplo 8.2

En la práctica, un ordenador digital no puede producir números al azar de forma perfecta y, estrictamente hablando, la función **RND** genera números **pseudoaleatorios.**

El generador de números pseudoaleatorios introduce una relación entre el argumento **X** y el número aleatorio seleccionado por la función, aunque, desgraciadamente, esa relación depende del ordenador. La versión del lenguaje de referencia de la función **RND**, que es la que se explica y utiliza en este libro, se usa en una gama bastante amplia de microordenadores.

Si el argumento es negativo, el generador de números aleatorios produce un número concreto que depende del valor del argumento. El número usado de esta forma se denomina a veces **semilla** del generador aleatorio. En otras palabras: **RND(−1)** producirá siempre el mismo número aleatorio sobre el mismo ordenador.

Una vez generado un número al azar a partir de una semilla, la función procede a generar una **secuencia** de números aleatorios, cada uno de los cuales depende del generado con anterioridad. Si el argumento es positivo, la función **RND** produce el siguiente número al azar de la secuencia; es decir, **RND(1)** da lugar a la aparición del siguiente número aleatorio de la secuencia.

Si hace falta el número al azar anterior de la secuencia, se utiliza como argumento de la función el valor cero. Es decir, **RND(0)** repite el número aleatorio que acaba de generarse.

La función **RND** tiene varias aplicaciones, como simulación, juegos y creación artística con ordenador.

8.9

Programa ejemplo 8.2

Un empleado debe recorrer todos los días para ir al trabajo el siguiente trayecto:

	Duración
Paseo hasta la estación del metro	3 minutos.
Espera en la estación	Entre 0 y 5 minutos, distribuidos uniformemente.
Trayecto en tren	Entre 7 y 10 minutos, distribuidos uniformemente.
Paseo desde la estación del metro	4 minutos.

El objetivo del programa es simular uno de tales trayectos.

Método

La función **RND** se utiliza para simular las partes variables del trayecto. El valor de la misma es un número comprendido entre 0 y 1 y uniformemente distribuido. Si este número se multiplica por 5, se obtiene un número entre 0 y 5 distribuido uniformemente; de la misma manera, si se multiplica por 3 y se le suma 7, se obtiene un número entre 7 y 10 uniformemente distribuido.

Para lanzar la función **RND** se introduce un número al principio del programa.

Variables

T1 Tiempo de camino hasta la estación.
T2 Tiempo de espera en la estación.
T3 Tiempo de viaje en tren.
T4 Tiempo de camino de la estación al trabajo.
T Tiempo total.
N Semilla para el generador de números aleatorios.
X Número aleatorio inicial.

Diagrama de flujo

Los pasos del programa se ilustran en la figura 8.3.

Programa

```
100 REM PROGRAMA EJEMPLO 8.2
105 REM SIMULADOR DE UN VIAJE
110 REM
115 PRINT "SIMULADOR DE UN VIAJE"
120 PRINT
125 PRINT "INTRODUZCA UN NUMERO POSITIVO PARA"
130 PRINT "INICIALIZAR EL GENERADOR DE NUMEROS
          ALEATORIOS"
135 INPUT N
140 LET X = RND(-N)
145 REM
200 LET T1 = 3
205 LET T2 = 5*RND(1)
210 LET T3 = 7 + 3*RND(1)
215 LET T4 = 4
220 LET T = T1 + T2 + T3 + T4
225 PRINT "TIEMPO DEL VIAJE"; T;"MINUTOS"
230 END
```

Puntos de interés

- En la línea 140 se "siembra" un número en el generador de números aleatorios.
- Las líneas 200 a 225 se han escrito de forma que el programa pueda modificarse para simular los trayectos de varias jornadas.

8.10

SGN(X)

La función de signo **SGN** se comporta como sigue:

SGN(X) = +1 si **X** es positivo.
SGN(X) = 0 si **X** es cero
SGN(X) = −1 si **X** es negativo

No hay limitaciones especiales respecto al intervalo de valores que puede adoptar el argumento de esta función.

8.11

SQR(X)

La función raíz cuadrada **SQR** calcula la raíz cuadrada positiva de su argumento. Por ejemplo, **SQR(16)** = 4 y **SQR(5)** = 2,236. El argumento de esta función ha de ser un número positivo o el cero.

8.12

SIN(X)

La función **SIN** calcula el seno de su argumento. Este se expresa en **radianes.** En la mayor parte de los casos, es más cómodo medir los ángulos en grados, que se reducen a radianes con arreglo a las siguientes fórmulas:

180 grados = 3,14159 radianes

$$\text{por tanto, 1 grado} = \frac{3{,}14159}{180} \text{ radianes}$$

$$\text{y } \mathbf{D} \text{ grados} = \frac{3{,}14159}{180} \times \mathbf{D} \text{ radianes.}$$

En otras palabras: la instrucción que asigna a la variable **S** el seno de un ángulo **D** medido en grados es

```
LET S = SIN(3.1415926 * D/180)
```

No hay limitaciones especiales respecto a los valores que puede adoptar el argumento de la función **SIN**.

Las funciones trigonométricas **SIN**, **COS**, **TAN** y **ATN** tienen muchas aplicaciones: navegación, agrimensura, diseño arquitectónico y de ingeniería, proyección de volúmenes tridimensionales en el plano, etcétera.

8.13

COS(X)

La función **COS** calcula el coseno de su argumento. Este se expresa en radianes. Véase la sección 8.12 respecto a la reducción de grados a radianes. Los valores válidos para el argumento no tienen limitaciones especiales.

8.14

TAN(X)

La función **TAN** calcula la tangente de su argumento expresado en radianes. Véase en la sección 8.12 un método de reducción de grados a radianes.

El valor de la función tangente es indefinido para ángulos de 90, 270, 450, etc., grados, y para ángulos negativos de la misma magnitud. Por tanto, estos valores no deben utilizarse como argumentos de dicha función.

8.15

ATN(X)

La función **ATN** calcula el ángulo cuya tangente es igual a su argumento. El ángulo se expresa en radianes. De esta definición se deduce que la función **ATN** es inversa de la función tangente.

En la mayor parte de las versiones del Basic, los ángulos calculados por esta función se encuentran entre $-1{,}5707$ y $+1{,}5707$ radia-

nes, intervalo equivalente al que hay entre -180 y $+180$ grados. Para reducir a grados el valor de la función **ATN** se sigue el método ya expuesto en la sección 8.12.

8.16
TAB(X)

La función de tabulación **TAB** se describió en la sección 3.8.

8.17
Programa ejemplo 8.3

El objetivo de este programa es producir una tabla de senos, cosenos y tangentes de todos los ángulos comprendidos entre 0 y 90 grados, con una precisión de tres decimales.

Método

Una vez creados los encabezamientos, el cuerpo principal del programa ha de estar obviamente constituido por un bucle controlado por contador. No se especifica el intervalo entre ángulos, por lo que se fijará en 1 grado. Dentro del bucle, los pasos a realizar son los siguientes:

Calcular el seno, el coseno y la tangente del ángulo reducido de grados a radianes conforme el procedimiento recogido en la sección 8.12.

Redondear los valores a tres cifras decimales.

Imprimir el ángulo y los valores redondeados de seno, coseno y tangente.

Debido a que la tangente no está definida para un ángulo de 90 grados, el bucle se repite desde 0 hasta 89 grados. El ángulo recto y su seno (1,000) y coseno (0,000) se imprimen por separado.

El redondeo a tres decimales se realiza mediante la función **INT** de la siguiente manera:

Multiplicar por 1.000.

Redondear al entero más próximo (véase sección 8.6).

Dividir por 1.000

La siguiente instrucción Basic redondeará la variable **J** a tres decimales:

```
LET K = INT(1000*J + 0.5)/1000
```

Diagrama de flujo

Se ilustra en la figura 8.4.

Variables

D Angulo en grados y contador del bucle.
R Angulo en radianes.
S Seno.
C Coseno.
T Tangente.

Programa

```
100 REM PROGRAMA EJEMPLO 8.3
105 REM IMPRIMIR TABLA DE SENO, COSENO, TANGENTE
110 REM
200 PRINT "ANGULO"; TAB(10); "SENO"; TAB(20);
          "COSENO"; TAB(30); "TANGENTE"
205 PRINT
210 FOR D = 0 TO 89
215 LET R = 3.14159*D/180
220 LET S = INT(SIN(R)*1000 + 0.5)/1000
225 LET C = INT(COS(R)*1000 + 0.5)/1000
230 LET T = INT(TAN(R)*1000 + 0.5)/1000
235 PRINT D ; TAB(10);S ; TAB(20);C ; TAB(30);T
240 NEXT D
245 PRINT 90; TAB(10); 1.000; TAB(20);0.000
250 END
```

Puntos de interés

- La función **TAB** se utiliza para alinear las cifras de la tabla con los encabezamientos.
- Los ángulos en grados se reducen primero a radianes, que se utilizan en las funciones seno, coseno y tangente.

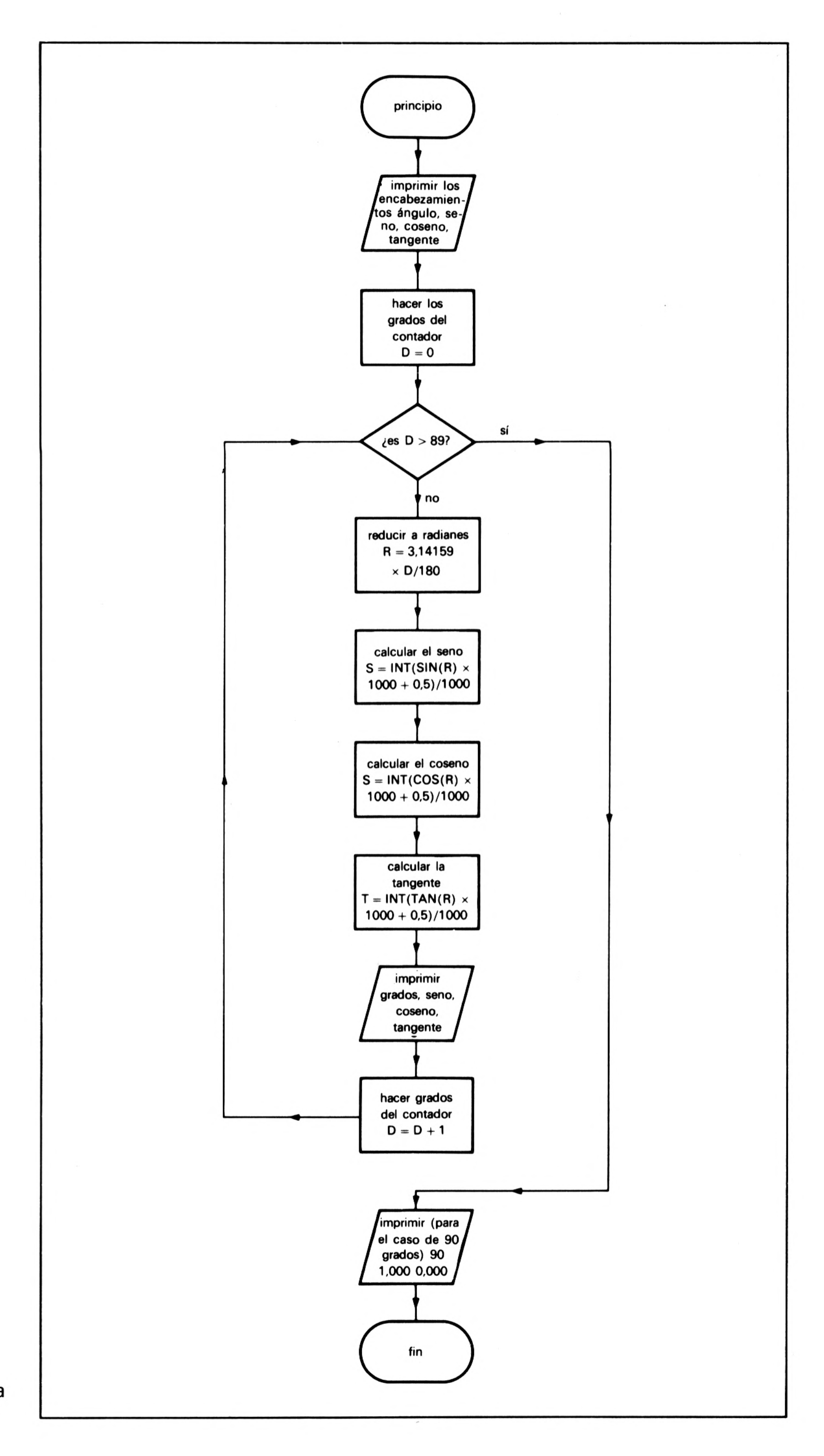


Figura 8.4
Diagrama de flujo del programa ejemplo 8.3

8.18

Funciones definidas por el usuario

Además de las funciones normalizadas expuestas en las secciones anteriores, el Basic permite la creación, con algunas limitaciones, de una serie de **funciones definidas por el usuario.** Estas funciones se nombran y definen al comienzo del programa o el segmento de programa en los que van a utilizarse. En las próximas secciones veremos cómo se definen y emplean.

8.19

Definición de una función

En Basic, para definir una función se usa la instrucción **DEF**. Esta instrucción da nombre a la función y proporciona la fórmula para evaluarla.

El nombre de cualquier función definida por el usuario debe empezar por las letras **FN**, a las que sigue una tercera y el signo **$** si la función tiene valor alfanumérico. He aquí algunos ejemplos de nombres de función correctos:

```
FNA,      FNJ$,     FNX.
```

Estos otros ejemplos corresponden a nombres incorrectos:

```
FN1,      FNA3,     FNSQUARE,  FN$.
```

En la instrucción **DEF**, el nombre de la función va seguido por un **argumento ficticio,** que también se utiliza en la fórmula de evaluación de la función. Así, la función **FNC**, que produce el cuadrado de su argumento, se definiría de la siguiente manera:

```
DEF FNC(A) = A*A
```

La variable **A** es el argumento supuesto.

La finalidad del argumento supuesto es indicar la relación existente entre el argumento y el valor de la función. El empleo de una variable como argumento supuesto es independiente de cualquier otro uso que pudiera darse a la misma variable en el resto del programa.

La instrucción **DEF** puede contener una función normalizada en

la fórmula. Por ejemplo, la función **POSN** mencionada al principio de este capítulo puede definirse haciendo uso de la función **ASC**:

```
DEF FNP(X$) = ASC(X$) - 64
```

El nombre en Basic de la función **POSN** lo hemos definido como **FNP** y sirve para contar la posición de cualquier letra del alfabeto.

8.20

Uso de una función definida por el usuario

Una vez definida, una función se utiliza como cualquiera de las normalizadas. El argumento no tiene por qué ser la misma variable que la utilizada en la instrucción **DEF**.

Cuando se usa una función definida, el argumento supuesto empleado en la instrucción **DEF** se sustituye por el argumento real, que se emplea para evaluarla. Por ejemplo, si **B$** tiene el valor "**J**" y **FNP** es la función de posición ya definida, la instrucción

```
LET A = FNP(B$)
```

asignará el valor 10 a la variable **A**, porque la **J** es la décima letra del alfabeto.

8.21

Programa ejemplo 8.4

El objetivo de este programa es simular un turno de una sencilla partida de dados para dos jugadores. Cada uno lanza un dado: si la puntuación es seis, tira otra vez y suma la puntuación obtenida a la existente para ese mismo turno.

Método

Para simular la tirada se usa una función definida por el usuario. El valor de esa función es un entero aleatorio comprendido entre 1 y 6 según una distribución uniforme. Este número se obtiene a partir de una función **RND** de la siguiente forma:

- La función **RND** genera un número aleatorio comprendido entre 0 y 1.

- La multiplicación de ese valor por 6 da lugar a un número comprendido entre 0 y 6.
- El redondeo a la parte entera da lugar a un número entero comprendido entre 0 y 5, porque la probabilidad de obtener exactamente 6 es tan baja que puede ignorarse.
- Basta ahora sumar 1 para obtener un entero aleatorio comprendido entre 1 y 6.

La fórmula de la función de tirada es la siguiente:

```
DEF FNT(X) = INT(RND(X)*6) + 1
```

Observe que la función tiene un argumento, que sirve para activar el generador de números aleatorios, según se discutió en la sección 8.8.

La parte del programa que simula el turno de un jugador puede diseñarse como un bucle de "repetición hasta":

Poner la puntuación total a cero.
Repetir el proceso:

tirar un dado
sumar la puntuación a la puntuación total.

Seguir hasta que la puntuación no sea seis.

El programa está escrito en dos módulos. El primero define la función de tirada y solicita la introducción de un número para sembrar el generador de números aleatorios. El segundo simula el turno de cada uno de los jugadores.

Las puntuaciones se almacenan en una matriz de dos elementos y la parte del programa correspondiente a cada uno de los jugadores se repite mediante un bucle **FOR ... TO**.

Variables

T(2)	Tanteos obtenidos por cada jugador en un turno.
X	Argumento supuesto de la definición de la función.
D	Tanteo obtenido en una puntuación concreta.
K	Contador del bucle.
I	Número que activa el generador de números aleatorios.

Diagrama de flujo

Se ilustra en la figura 8.5.

Programa

```
100 REM PROGRAMA EJEMPLO 8.4
105 REM JUEGO DE DADOS
110 REM
200 REM MODULO DE INICIALIZACION
205 DIM T(2)
210 DEF FNT(X) = INT(RND(X)*6) + 1
215 PRINT "JUEGO DE DADOS"
220 PRINT
225 PRINT "INTRODUZCA UN NUMERO POSITIVO PARA
           INICIALIZAR"
230 PRINT "EL GENERADOR DE NUMEROS ALEATORIOS" ;
```

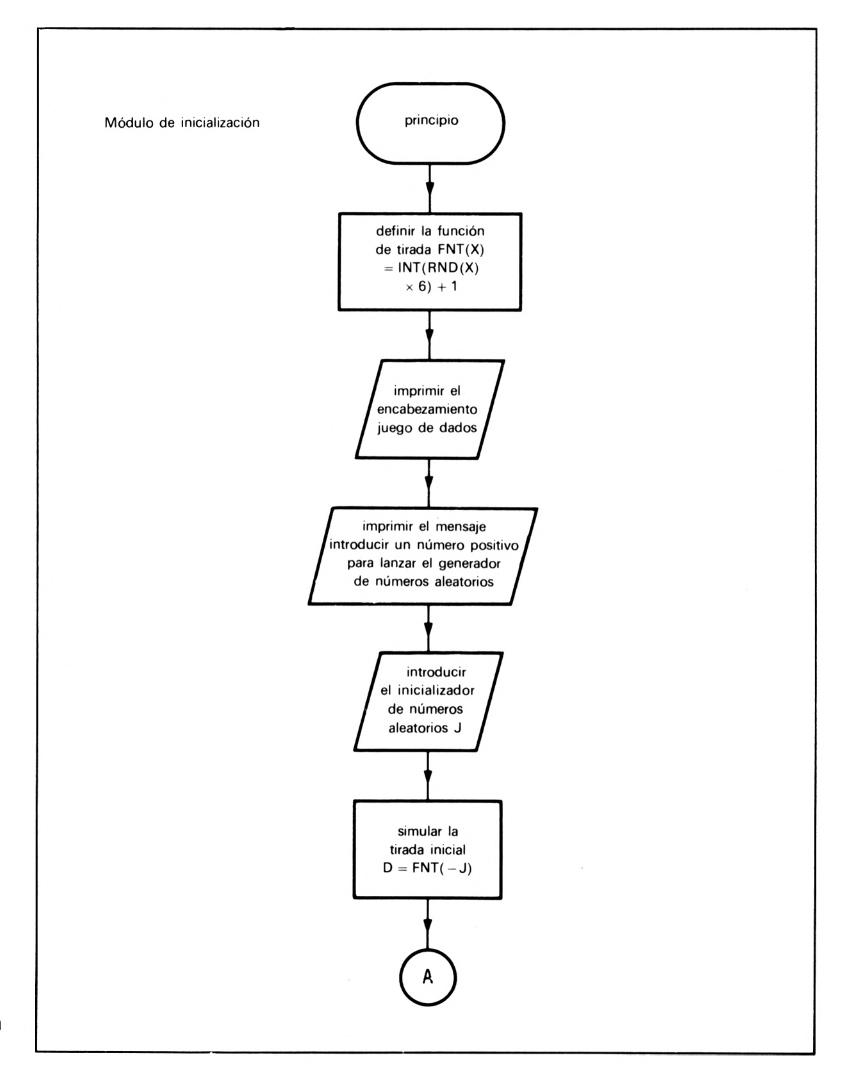
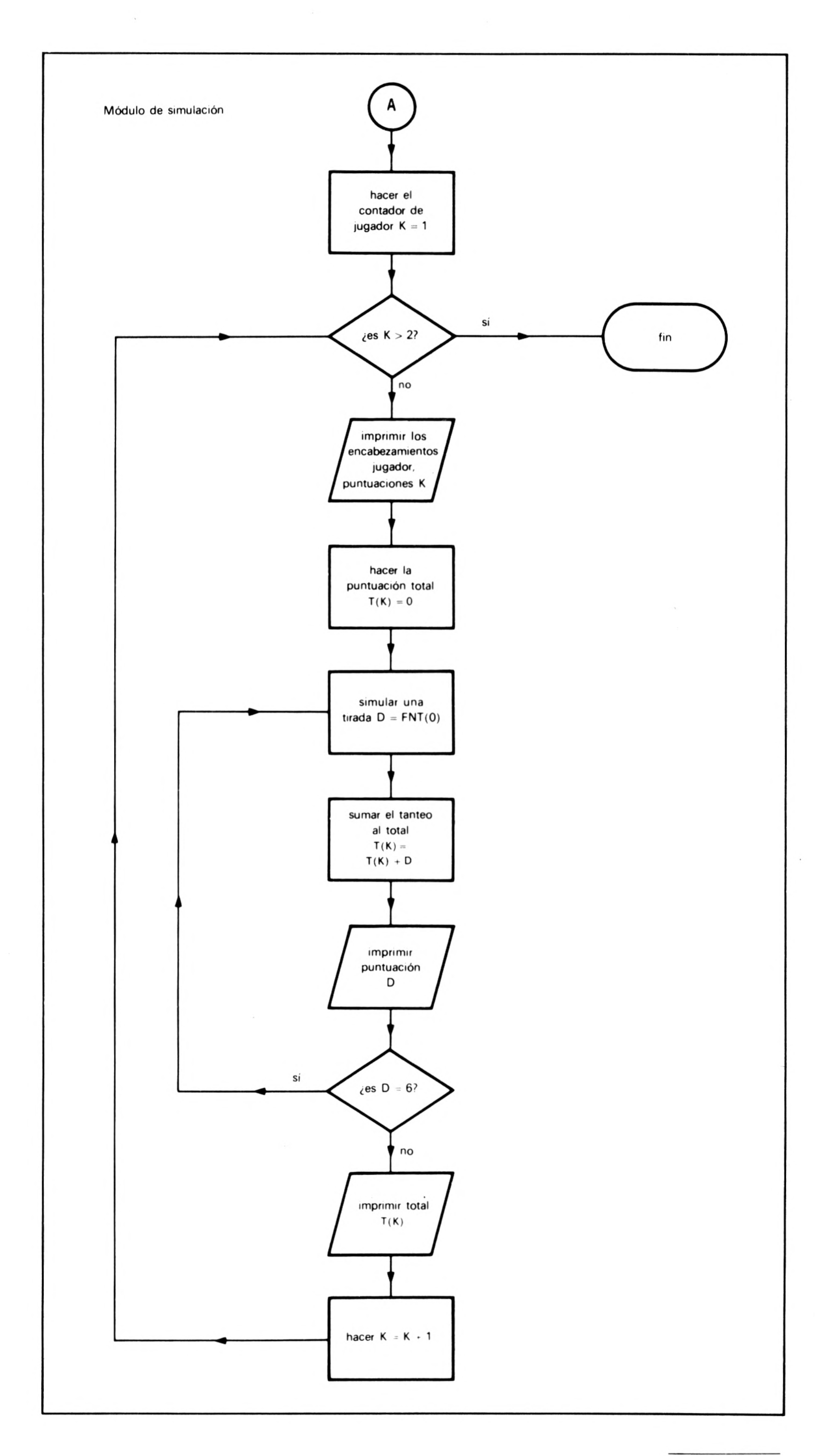


Figura 8.5
Diagrama de flujo del programa ejemplo 8.4



Módulo de simulación
A
hacer el contador de jugador K = 1
¿es K > 2?
si
fin
no
imprimir los encabezamientos jugador, puntuaciones K
hacer la puntuación total T(K) = 0
simular una tirada D = FNT(0)
sumar el tanteo al total T(K) = T(K) + D
imprimir puntuación D
¿es D = 6?
si
no
imprimir total T(K)
hacer K = K + 1

```
235 INPUT J
240 LET D = FNT(-J)
245 REM
300 REM SIMULADOR DE UNA TIRADA
305 FOR K = 1 TO 2
310 PRINT "JUGADOR"; K;
315 PRINT "PUNTOS:";
320 LET T(K) = 0
325 LET D = FNT(1)
330 LET T(K) = T(K) + D
335 PRINT D;
340 IF D = 6 THEN 325
345 PRINT
350 PRINT "TOTAL:"; T(K)
355 NEXT K
360 END
```

Puntos de interés

- El contador **K** se usa como índice de la matriz de tanteos.
- La condición "repetir hasta que" (línea 340) se ha escrito como condición para continuar la repetición.
- El programa se ha diseñado de manera que pueda modificarse en el sentido de incluir más de dos jugadores y más de un turno.

8.22

Otras características de las funciones definidas por el usuario

Algunas versiones del Basic permiten al usuario definir funciones con más de un argumento. En otras versiones, la definición puede escribirse a lo largo de varias líneas del programa. Como estas opciones dependen del tipo de ordenador, no se han incluido en el lenguaje de referencia.

8.23

Conclusión

Este capítulo constituye una introducción bastante completa al tema de las funciones tal como se ejecutan en Basic. Los puntos más importantes son:

- Una función puede considerarse como una asociación entre dos magnitudes.

- Una función tiene un **nombre.** Las funciones Basic normalizadas tienen nombres de tres letras, como **INT**; las definidas por el usuario empiezan siempre por las letras **FN** a las que sigue una tercera. Al final de estas tres letras se añade el signo **$** si la función tiene valor alfanumérico.
- Se llama **argumento** de una función a la cantidad sobre la que opera.
- El resultado de aplicar una función a un argumento es el **valor** de dicha función.
- Pueden definirse funciones por medio de la instrucción **DEF**, que debe situarse en el programa antes de que dicha función vaya a utilizarse.

8

Ejercicio

1. Defina brevemente los siguientes términos: función, función normalizada, función definida por el usuario, argumento, valor de una función, función inversa.
2. ¿Puede el argumento de una función adoptar cualquier valor? Apoye la respuesta en algunos ejemplos.
3. Escriba una instrucción **LET** con una o varias de las funciones normalizadas para realizar las operaciones siguientes:
   a) Asignar a la variable **X** la diferencia positiva entre las variables **A** y **B**.
   b) Asignar a la variable **X** el valor de la **Y** redondeado a una cifra decimal.
   c) Asignar a la variable **M** un entero aleatorio comprendido entre 1 y 100.
   d) Asignar a la variable **X** el valor 0 si la variable **Y** es cero y el valor 1 si la variable **Y** no es cero.
   e) Asignar a la variable **H** la longitud de la hipotenusa de un triángulo rectángulo cuyos dos lados restantes son **A** y **B**.
   f) Asignar a la variable **D** el ángulo en grados cuya tangente es **T**.
4. En un entorno ideal, la población de una especie crece con arreglo a la fórmula

   $$p = p_0 e^{(\log_e(2)t/T)}$$

   donde t es el tiempo, p_0 es la población en $t = 0$, p es la población actual y T el tiempo que la población tarda en duplicarse.

   Modifique el programa ejemplo 8.1 con el fin de simular el crecimiento de una población específica a partir de un valor inicial y hasta que alcance diez veces ese valor. Como entrada se usará el tiempo que tarda la población en duplicarse y a la salida se obtendrán los tamaños de la población a intervalos de tiempo apropiados.
5. Modifique el programa ejemplo 8.2 para simular varios trayectos. Obtenga a la salida el tiempo de cada trayecto y calcule y obtenga también a la salida la duración media del desplazamiento. Si lo desea, puede también calcular la desviación típica de los tiempos de trayecto.

6. Modifique el programa ejemplo 8.4 para simular:
   a) Un turno de una partida con 5 jugadores.
   b) Diez turnos de una partida con 5 jugadores. Calcule para cada jugador el tanteo total acumulado.
7. La posición de un punto en un espacio bidimensional queda definida por un vector. Así,

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

describe un punto de coordenadas 1 y 3, respectivamente. Una matriz puede describir un movimiento o **transformación** del punto. Por ejemplo, la rotación en el sentido de las agujas del reloj de un punto a lo largo de un ángulo de D grados queda reflejada por la matriz

$$\begin{bmatrix} \cos D & \operatorname{sen} D \\ -\operatorname{sen} D & \cos D \end{bmatrix}$$

El vector que describe el punto transformado se obtiene multiplicando el vector original por la matriz de la transformación. En este caso sería:

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix} \times \begin{bmatrix} \cos D & \operatorname{sen} D \\ -\operatorname{sen} D & \cos D \end{bmatrix}$$

   a) Escriba un módulo de programa que, dados el vector que representa un punto y un ángulo de rotación, produzca el vector del punto transformado.
   b) Modifique el módulo de la parte *a*) para convertirlo en un programa completo con las correspondientes rutinas de entrada y salida; *o bien*, escriba una aplicación completa de transformación estructurada como el programa ejemplo 7.2. Utilice un libro de matemáticas para obtener las fórmulas de las demás transformaciones.
8. La función generadora de números aleatorios **RND** puede utilizarse tal como se sugiere a continuación para simular la probabilidad de que un suceso al azar ocurra o no ocurra.

   La probabilidad de que un suceso aleatorio ocurra viene determinada por un número P de valor comprendido entre 0 y 1. Si $P = 0$, el suceso no ocurrirá, y si $P = 1$ ocurrirá con toda seguridad; los valores intermedios corresponden a probabilidades situadas entre esos dos extremos.

   Dado un valor de P, la ocurrencia o no ocurrencia de un suceso se simula eligiendo un número aleatorio R comprendido entre 0 y 1 por medio de la función **RND**. Si R es menor o igual a P, es que el suceso ha ocurrido; si R es mayor que P, el suceso no ha ocurrido. Esta misma técnica es aplicable a gran diversidad de casos de simulación, como por ejemplo el siguiente:

   La llegada de llamadas telefónicas a una centralita puede simularse aproximadamente de la siguiente forma:

   > Durante un período concreto de un minuto, la probabilidad de que llegue una llamada es 0,3. La probabilidad de que lleguen dos en el mismo minuto se desprecia.
   > Con la técnica de simulación esbozada más arriba, simule una hora de trabajo de la central.

Obtenga a la salida los minutos en que se reciben llamadas y el número medio de llamadas por minuto.

9. Defina una función que, cuando reciba como argumento una letra mayúscula, produzca como resultado la minúscula correspondiente. Defina igualmente la función inversa.

10. Escriba las instrucciones necesarias para controlar un bucle mediante un contador que va de **A** a **B** en pasos de **C**, con independencia de que **B** sea o no mayor que **A**. (Sugerencia: utilice las funciones **SGN** y **ABS**.)

11. **Otros posibles programas:** pueden escribirse programas en los que intervengan funciones sobre muchos temas a nivel de BUP. Damos a continuación algunas sugerencias:

Física: balística, movimiento rotativo.
Química: velocidades de reacción.
Biología: crecimiento de organismos y de poblaciones.
Matemáticas: trigonometría, cálculo, geometría de coordenadas.


RESET
RETURN

9 Subprogramas

Estudiaremos en este capítulo los **subprogramas** y su realización en Basic. Discutiremos también la relación que hay entre los subprogramas y la estructura general del programa y el paso de datos a los subprogramas y desde ellos al resto del programa.

En los capítulos anteriores hemos estudiado técnicas pensadas para realizar en Basic operaciones específicas, técnicas que se han usado para crear partes o módulos de programas. Este capítulo tratará de la construcción de programas completos a partir de tales módulos.

Utilizaremos aquí todo el material de los capítulos anteriores y lo pondremos en relación con los próximos, sobre todo con el 11, que trata del diseño de programas, y con los que van del 18 al 21, dedicados a la programación de estructuras fundamentales de datos.

9.1 Naturaleza de los subprogramas

Como su propio nombre indica, un subprograma es una parte de un programa que ejecuta una tarea específica incluida dentro de la más general que realiza este último. El subprograma guarda con el

resto del programa, al que en este contexto se llama **programa principal,** una relación peculiar.

Cada vez que se necesita un subprograma, se le **llama** desde un punto del programa principal. Cuando termina, el control **vuelve** al punto del programa principal desde el que se efectuó la llamada. Un mismo subprograma puede llamarse desde varios puntos diferentes del programa principal. La figura 9.1 ilustra la relación que hay entre un subprograma y el programa principal.

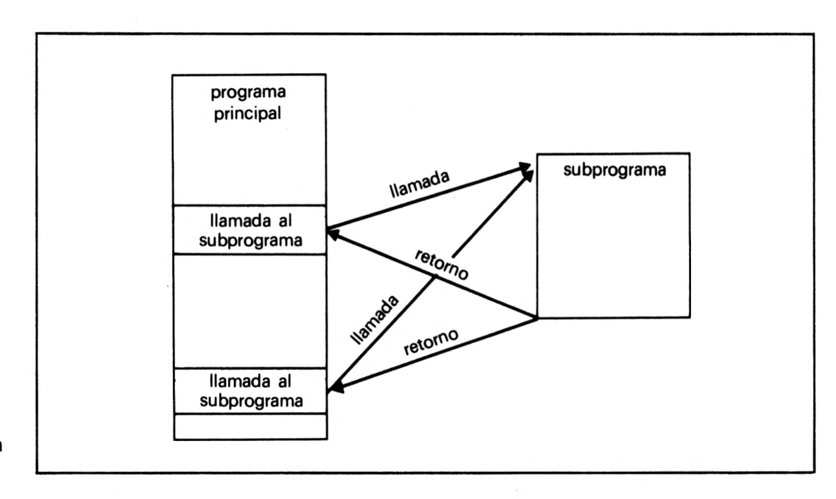


Figura 9.1
Relaciones entre el programa principal y un subprograma

Un subprograma puede también ser llamado desde otro. Algunos lenguajes de programación permiten que un subprograma se llame a sí mismo, opción que se conoce con el nombre de **recursión.** Sin embargo, la mayor parte de las versiones del Basic carecen de esta posibilidad.

Un subprograma puede considerarse simplemente como una porción de un código necesario en cierto número de puntos del programa, aunque se trata de un punto de vista un tanto limitado. Es más útil considerar los subprogramas como "ladrillos" constitutivos de la estructura general del programa. En este contexto, se conocen también como **procedimientos.** Este punto de vista se discute más detalladamente en el capítulo 11.

9.2 Los subprogramas en Basic

Por desgracia, en Basic el tratamiento de los subprogramas tiene bastantes carencias. Algunas de éstas se exponen en esta sección y otras se harán evidentes hacia el final del capítulo. Es preciso

subrayar que el diseño cuidadoso de los subprogramas es fundamental para superar las deficiencias propias del lenguaje.

Ninguna instrucción especial señala el inicio de un subprograma, aunque en todos los incluidos en este libro se emplea para ese fin una sentencia **REM** que establece el nombre y la finalidad del subprograma.

El final de un subprograma queda marcado por una instrucción **RETURN**. Un subprograma puede tener más de un punto de retorno. Veremos a continuación qué instrucciones sirven para llamar a un subprograma.

La instrucción **GOSUB** crea una bifurcación incondicional hacia un subprograma. A la palabra de instrucción sigue el número de línea en que se encuentra la primera instrucción del subprograma. Por ejemplo:

```
GOSUB 5000
```

transfiere el control al subprograma que empieza en la línea 5000.

La bifurcación condicional hacia un subprograma se crea mediante la construcción

IF <condición> **THEN GOSUB** <número de línea>

Si la condición se cumple, se produce la llamada al subprograma. Y tanto si se cumple como si no, el control vuelve a la instrucción siguiente a la **IF**.

La bifurcación múltiple hacia varios subprogramas se establece como sigue:

ON <expresión de control> **GOSUB** <número de línea> <número de línea> <número de línea>, etc.

Si la expresión de control, que puede ser una simple variable, toma el valor 1, se llama al primer subprograma, si el 2, al segundo, etc.

Un punto interesante es que en Basic las variables son comunes al programa principal y a todos sus subprogramas. Esto contrasta con lo que ocurre en muchos otros lenguajes de programación, en los que las variables son **locales,** es decir, propias de cada subprograma. Al escribir un subprograma hay que tener cuidado para no alterar inadvertidamente los valores de las variables del programa principal o de otros subprogramas. Este punto volverá a discutirse en este mismo capítulo.

9.3

Programa ejemplo 9.1

Este programa permite utilizar un ordenador como una sencilla calculadora capaz de realizar cálculos simples y "encadenados".

Método

Para diseñar este programa hay que fijarse primero en el funcionamiento de una calculadora. Para cálculos simples, la secuencia de acciones es

teclear un número
pulsar un signo de operación (+ – * o /)
teclear un número
pulsar =

Los cálculos secuenciales se efectúan así:

teclear un número
pulsar un signo de operación
teclear un número
pulsar un signo de operación..., etc., hasta llegar al último número
teclear el último número
pulsar =

En este caso, sea cual sea el operador pulsado, siempre se ejecuta la operación inmediatamente **anterior.** La tecla = se encarga de llevar a cabo la última operación.

Para trabajar con todas estas secuencias de operaciones, el programa utiliza dos espacios de almacenamiento llamados **Registro A** y **Registro B**. Cada operación se realiza sobre los dos números situados en dichos registros, y el resultado se sitúa en el **A**.

Además de las operaciones matemáticas, la calculadora necesita una función adicional llamada **reproducción,** que se limita a transferir al registro **A** el contenido del **B**. La utilidad de esta operación se hará evidente en seguida.

Las operaciones se identifican mediante el siguiente código de números:

Símbolo de la operación	*Número de control*	*Acción*
=	1	Copia el registro **B** en el **A**
+	2	A = A + B
—	3	A = A - B
*	4	A = A * B
/	5	A = A / B

Con estas operaciones, el control general de la calculadora se efectúa como sigue:

Llevar la operación anterior a copiar (código 1).
Repetir el proceso.

- Introducir un número en el registro **B**.
- Introducir una operación.
- Verificar la operación: si es válida, adjudicarle un número de código;
  en caso contrario, repetir el paso anterior.
- Efectuar la operación anterior y llevar el resultado al registro **A**.
- Igualar la operación anterior al código de la que acaba de introducirse.
- Presentar el contenido del registro **A**.

Hasta completar el cálculo.

Se emplean subprogramas para verificar los símbolos de operación y para efectuar cada una de éstas.

Variables

A Registro **A**.
B Registro **B**.
I$ Introducción de símbolo de operación.
C Código de la operación en marcha.
P Código de la operación previa.

Diagrama de flujo

La figura 9.2 ilustra el flujo de control del programa.

Programa

```
100 REM PROGRAMA EJEMPLO 9.1
105 REM CALCULADORA
110 REM
200 REM PROGRAMA PRINCIPAL : MODULO DE CONTROL
205 LET P = 1
210 INPUT B
215 INPUT I$
220 GOSUB 300 : REM VERIFICA LA OPERACION
225 IF C <> 0 THEN 240
```

```
230 PRINT "OPERACION INCORRECTA"
235 GOTO 215
240 ON P GOSUB 400, 450, 500, 550, 600
245 LET P = C
250 PRINT A
255 GOTO 210
260 REM FIN DEL PROGRAMA PRINCIPAL
300 REM SUBPROGRAMA QUE VERIFICA LA OPERACION
305 LET C = 0: REM LA OPERACION SE SUPONE NO
                        VALIDA
310 IF I$= "=" THEN LET C =1
```

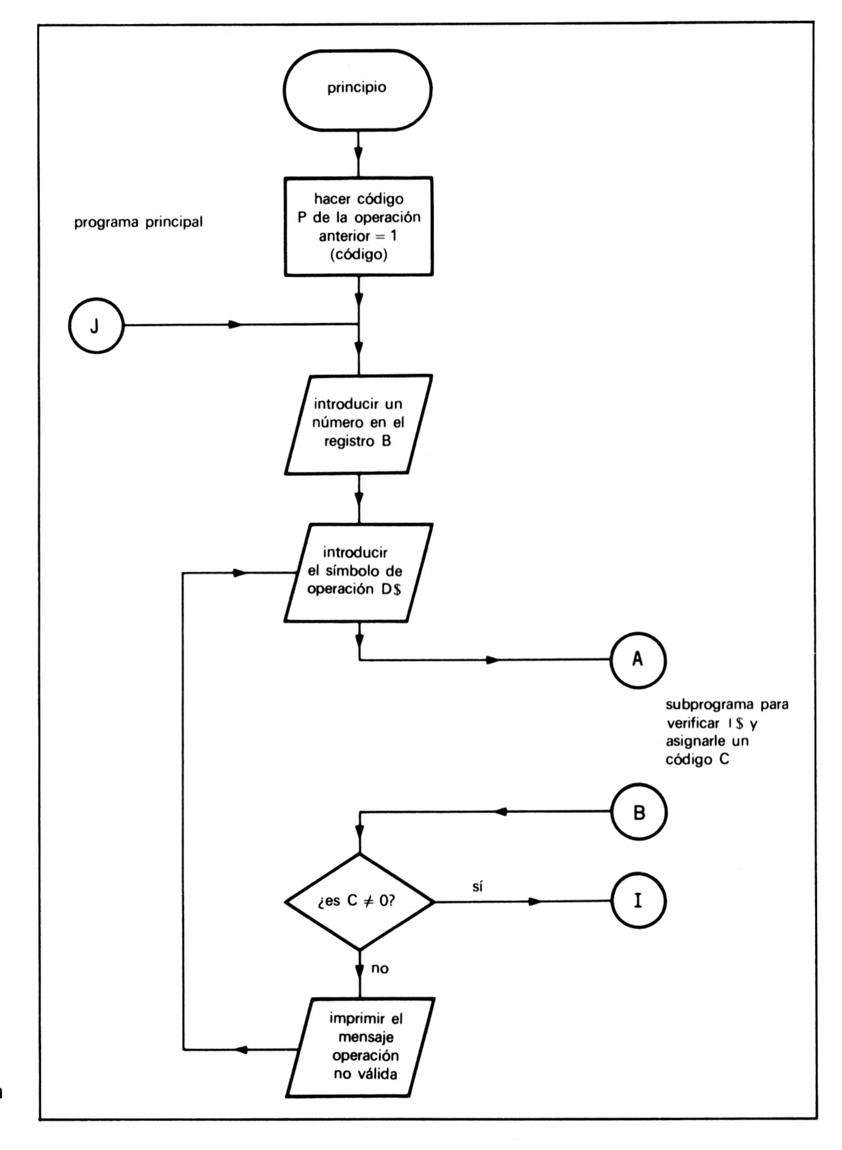
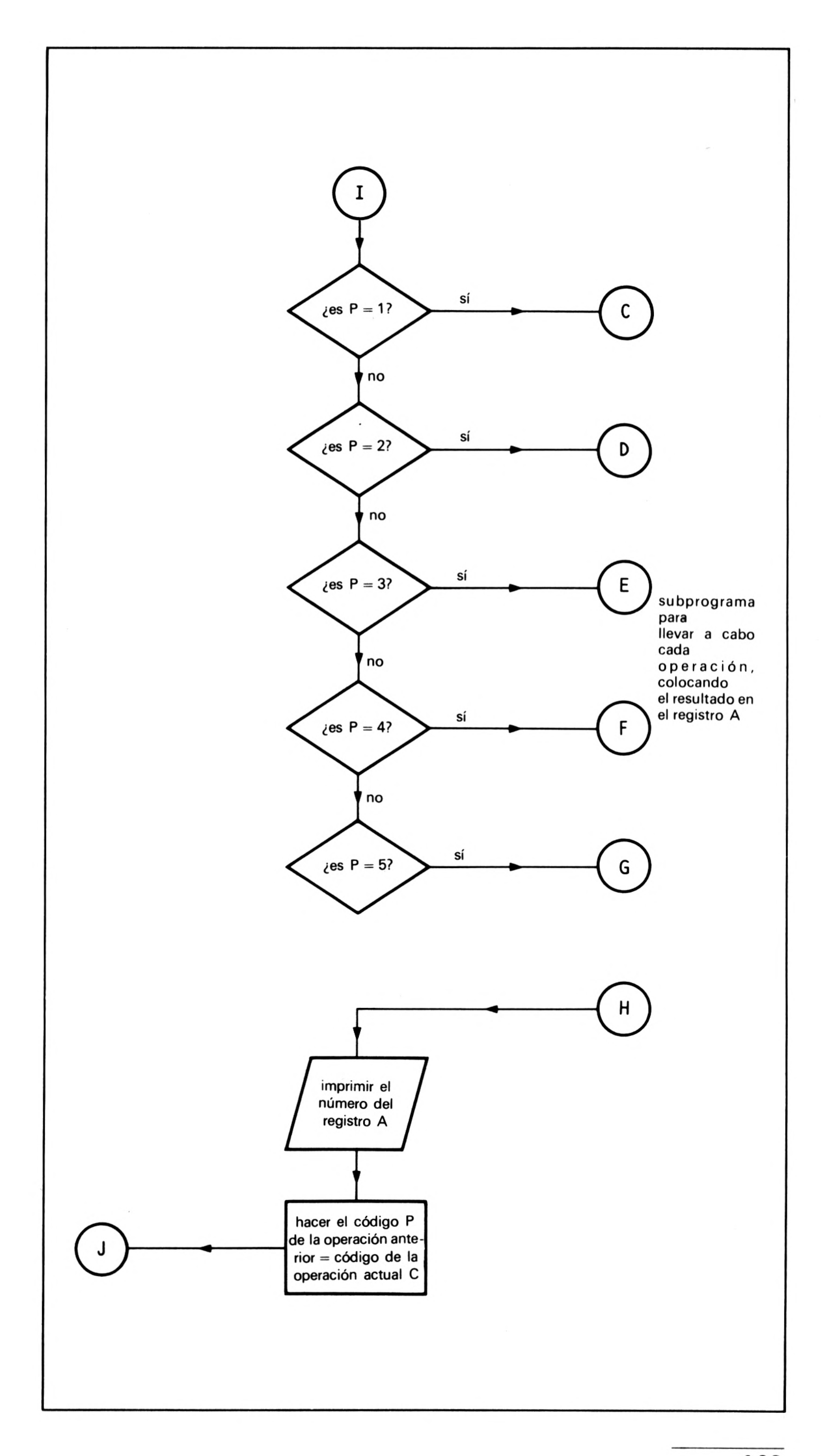
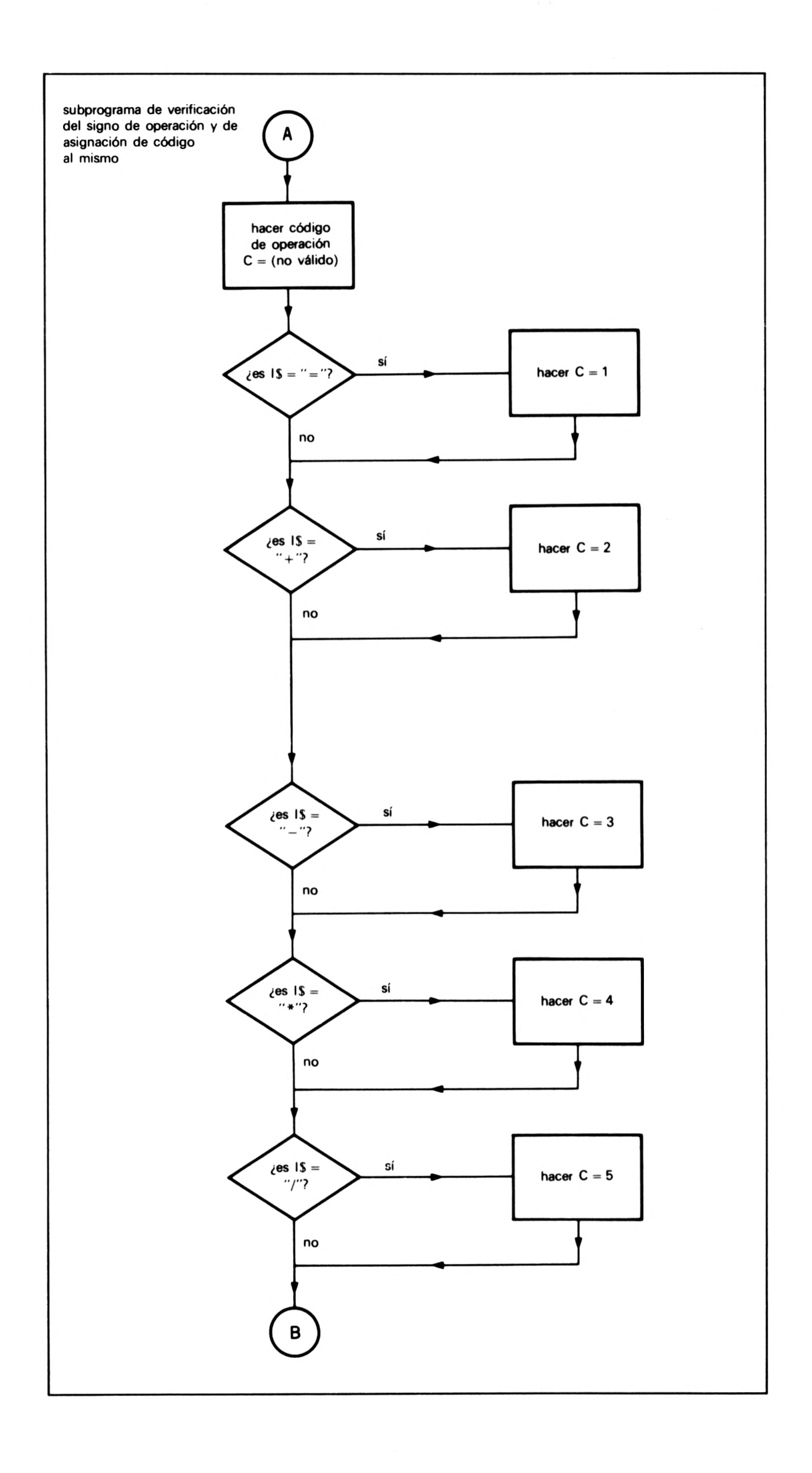
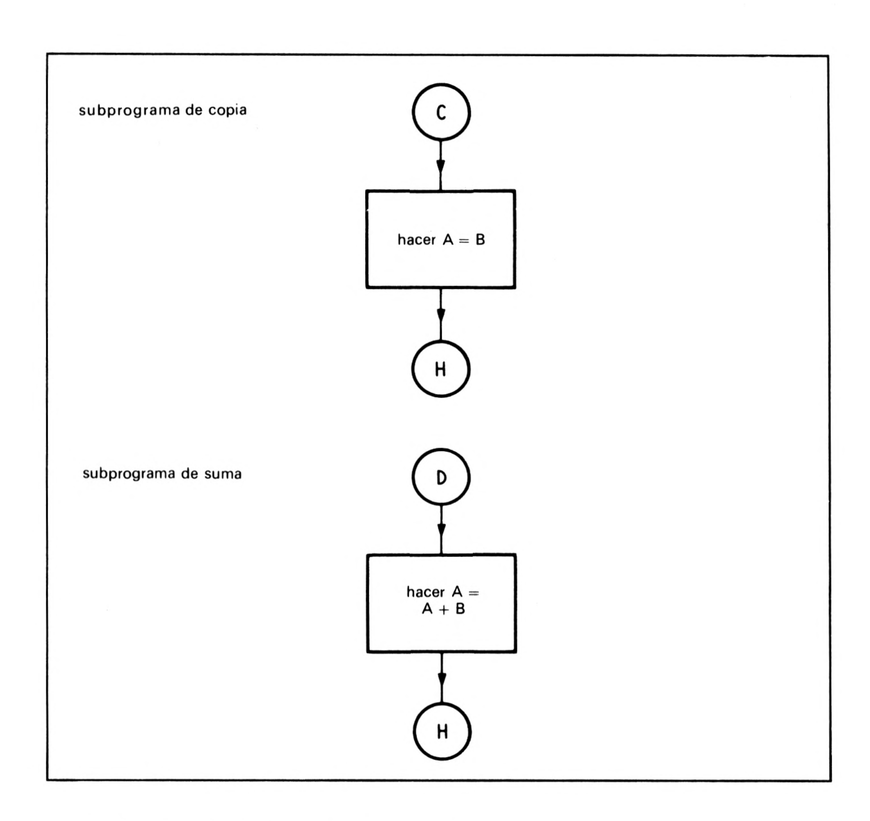


Figura 9.2
Diagrama de flujo del programa ejemplo 9.1

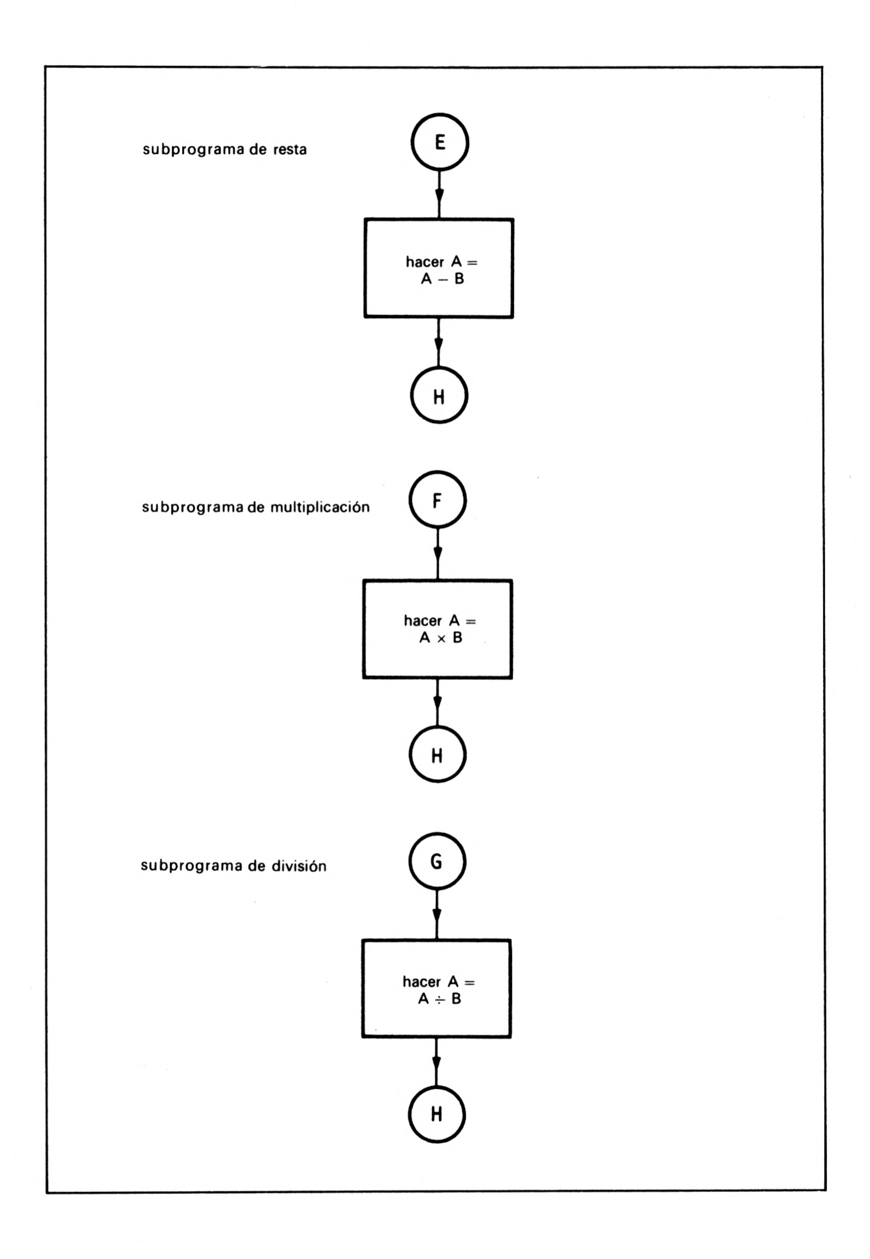

I
¿es P = 1?
sí
C
no
¿es P = 2?
sí
D
no
¿es P = 3?
sí
E
no
¿es P = 4?
sí
F
no
¿es P = 5?
sí
G
subprograma para llevar a cabo cada operación, colocando el resultado en el registro A
H
imprimir el número del registro A
hacer el código P de la operación anterior = código de la operación actual C
J



subprograma de verificación del signo de operación y de asignación de código al mismo
A
hacer código de operación C = (no válido)
¿es I$ = "="?
sí
hacer C = 1
no
¿es I$ = "+"?
sí
hacer C = 2
no
¿es I$ = "−"?
sí
hacer C = 3
no
¿es I$ = "*"?
sí
hacer C = 4
no
¿es I$ = "/"?
sí
hacer C = 5
no
B




```
315 IF I$= "+" THEN LET C =2
320 IF I$= "-" THEN LET C = 3
325 IF I$= "*" THEN LET C = 4
330 IF I$= "/" THEN LET C = 5
335 RETURN
400 REM SUBPROGRAMA QUE COPIA
405 LET A = B
410 RETURN
450 REM SUBPROGRAMA QUE SUMA
455 LET A = A + B
460 RETURN
500 REM SUBPROGRAMA QUE RESTA
505 LET A = A - B
510 RETURN
550 REM SUBPROGRAMA QUE MULTIPLICA
555 LET A = A * B
560 RETURN
600 REM SUBPROGRAMA QUE EVITA LA DIVISION POR O
605 IF B = O THEN PRINT "ERROR. SE HA DIVIDIDO
                         POR O. INTRODUZCA
                         NUEVOS DATOS":RETURN
610 REM SUBPROGRAMA QUE DIVIDE
615 LET A = A / B
620 RETURN
650 END
```



Puntos de interés

- La calculadora carece de la operación de borrado, que se ha omitido deliberadamente en aras de la sencillez del programa. Su inclusión está relacionada con uno de los ejercicios propuestos al final del capítulo.
- No hay forma de interrumpir la ejecución del programa. Este

comportamiento es similar al de una calculadora, que sólo se detiene cuando se desconecta.

- El subprograma de verificación empieza por suponer que la operación es incorrecta y hace un código igual a 0. Si el signo de la operación coincide con el de alguna de las instrucciones **IF**, el código cambia; en caso contrario, sigue siendo 0 al final del subprograma.
- En el subprograma de verificación se ha utilizado la versión perfeccionada de la instrucción **IF**.
- Cada uno de los subprogramas de este programa ejecuta una operación sencilla y claramente definida.
- Las líneas 600, 605 que imprimen un mensaje de error, en el caso de que se intente efectuar una división por 0.

9.4

Intercambio de datos entre el programa principal y los subprogramas

La finalidad de un subprograma es realizar una tarea específica, para lo que utiliza ciertos datos a partir de los que obtiene ciertos resultados. En general, este proceso implica la realización de una o más de las siguientes tareas:

Introducir un dato y transferir su valor al programa principal.
Recibir del programa principal el valor de un dato y presentar su valor a la salida.
Aceptar uno o más datos del programa principal, procesarlos y devolver los resultados a dicho programa principal.

En todos los casos puede verse que hay tráfico de datos entre el programa principal y el subprograma. En algunos casos, este tráfico se realiza utilizando en el subprograma la misma variable que en el programa principal; una variable usada de esta forma se califica de **global.** Esta técnica se utiliza en el programa ejemplo 9.1.

No obstante, y por diversas razones, suele ser preferible adjudicar la transferencia de datos a variables especiales que se llaman **parámetros.** La mayor parte de los lenguajes de programación disponen de recursos específicos para el empleo de parámetros, pero el Basic, lamentablemente, no está entre ellos.

En Basic, la ventaja principal de trabajar con parámetros es que éstos hacen al subprograma completamente independiente del programa principal (suponiendo que, aparte los parámetros, aquél no

tenga más nombres de variables en común con éste). Además de constituir un buen método de diseño de programas, el uso de parámetros evita los **efectos secundarios** que se producen cuando, inadvertidamente, un módulo de un programa afecta a los valores de otros. Además, si se trabaja con parámetros, puede escribirse un subprograma que sirva para varios programas diferentes.

En este libro, el uso de parámetros se abordará de la siguiente manera:

Antes de llamar a un subprograma, se asignan valores a todos los parámetros encargados de transferir datos al mismo.

Después de que el control ha sido cedido por el subprograma, todos los parámetros encargados de transportar datos desde el mismo reproducen sus contenidos en variables del programa principal.

De acuerdo con lo dicho, es preciso un grupo de instrucciones para transferir el control desde el programa principal a un subprograma. En ese grupo se encuentra la instrucción **GOSUB** y varias otras de asignación de parámetros.

No siempre es posible, ni deseable, que todas las variables de un subprograma sean completamente independientes de las del programa principal. En caso de que se compartan, se adopta la convención siguiente:

Ciertas variables de control son comunes al programa principal y a todos los subprogramas, y se llaman **globales.**

Otras son privativas de un subprograma específico, y se llaman **locales.**

9.5

Programa ejemplo 9.2

En estadística se usa muchas veces una función llamada **factorial** que se define como sigue:

el factorial de 0 es 1

el factorial de cualquier entero n mayor que 0 es

$$n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 \text{ y se escribe } n!$$

La función factorial no se define para números negativos.

Como la función factorial no puede evaluarse dentro de una sola instrucción **DEF**, es preciso escribir para ella un subprograma. El subprograma evalúa el factorial de un número si es positivo o cero y da como resultado cero si el número es negativo.

Se usan dos parámetros, uno para el número (transferido al subprograma) y otro para el valor de la función factorial (transferido desde el subprograma). El programa principal se diseña y se escribe a continuación, para ilustrar cómo se usa el subprograma.

Método

Primero se verifica el número para ver si es negativo, cero o positivo. Si es negativo, el valor de la función factorial se hace igual a cero. Si es cero, se hace igual a 1. Si es positivo, se utiliza un bucle **FOR ... TO** para acumular el valor del factorial con arreglo al siguiente algoritmo:

Fuera del bucle, el factorial se pone a 1.
Se utiliza un contador de bucle, que va desde 1 hasta el valor del número.
Dentro del bucle, el factorial se multiplica por el valor actual del contador. De esta forma se va elaborando el producto

$$1 \times 2 \times 3 \times \cdots \times (n - 1) \times n$$

Variables

Parámetros:	**N**	Número (transferido al subprograma).
	F	Factorial (transferido por el subprograma).
Variable local:	**K**	Contador del bucle, utilizada dentro del subprograma.

Diagrama de flujo

El flujo de control del subprograma se ilustra en la figura 9.3.

Subprograma

```
1000 REM SUBPROGRAMA FACTORIAL
1005 REM PARAMETROS
1010 REM N : NUMERO
1015 REM F : FACTORIAL
1020 REM
```

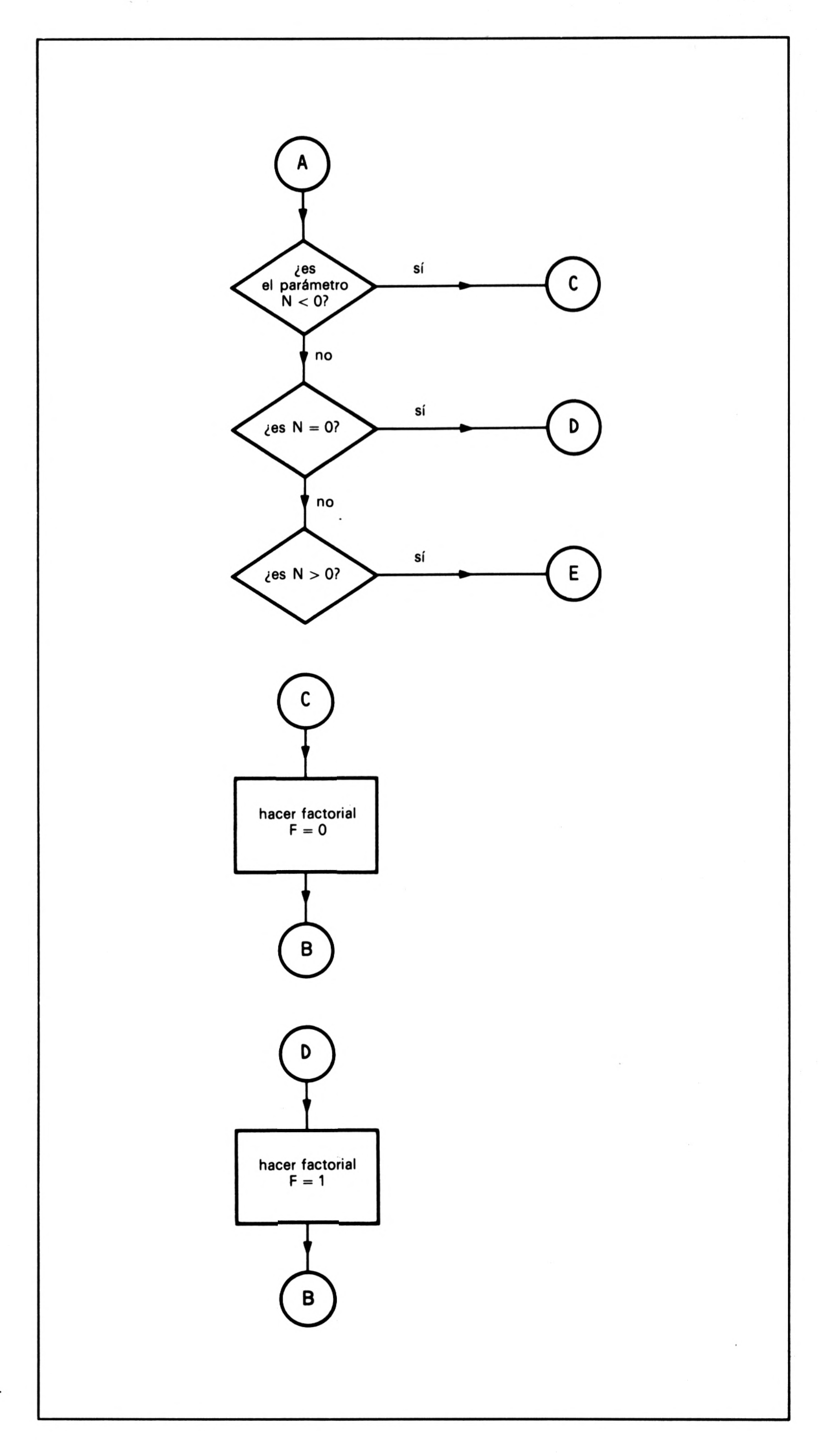
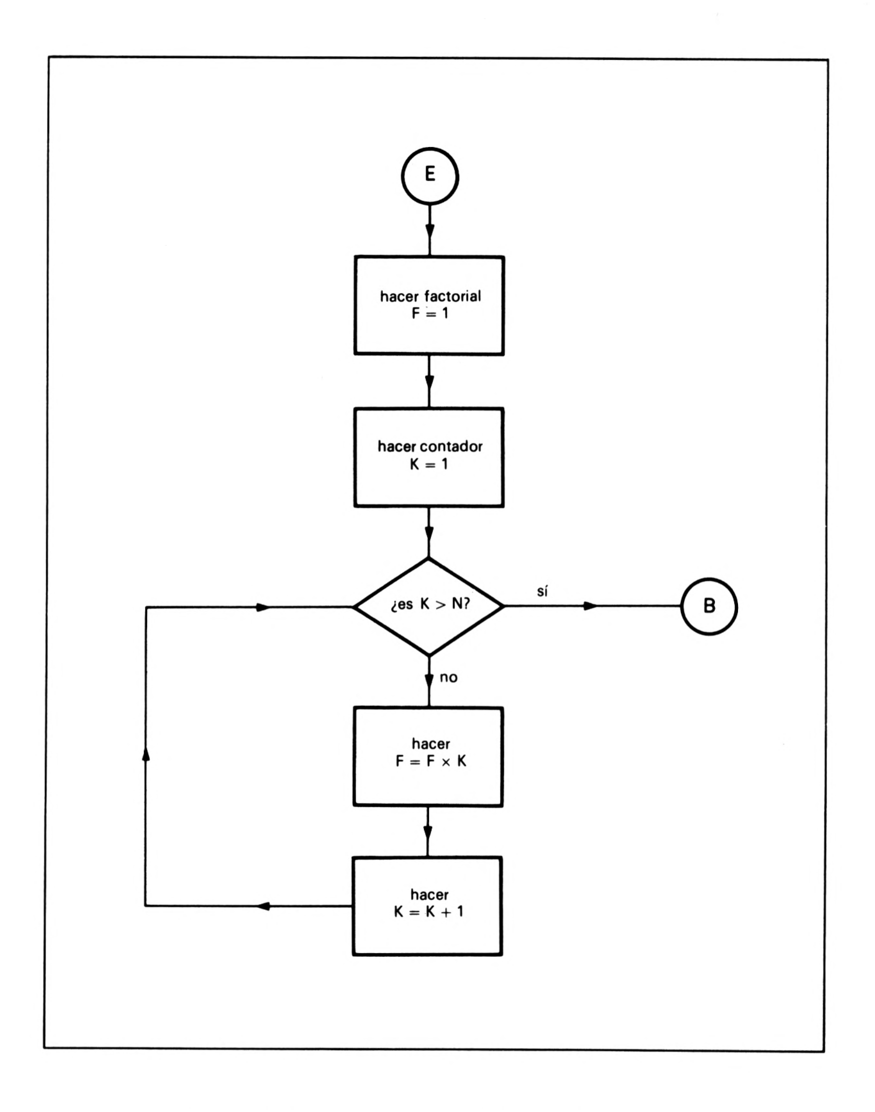


Figura 9.3
Diagrama de flujo del subprograma del programa ejemplo 9.2




```
1025 ON SGN(N) + 2 GOTO 1100, 1200, 1300
1030 REM
1100 REM VALOR NEGATIVO DE N
1105 LET F = 0
1110 RETURN
1200 REM N VALE 0
1205 LET F = 1
1210 RETURN
1300 REM VALOR POSITIVO DE N
1305 LET F = 1
1310 FOR K = 1 TO N
1315 LET F = F * K
1320 NEXT K
1325 RETURN
```

Puntos de interés

- En la instrucción **ON ... GOTO** de la línea 1025, la expresión de control es **SGN(N) + 2**, que toma el valor 1 si **N** es negativo, el 2 si es cero y el 3 si es positivo.
- Este subprograma no verifica si **N** es entero. De todas formas, si no lo fuese, calcularía el factorial de la parte entera del mismo.

Programa principal

Este programa acepta repetidamente la introducción de un número y presenta como salida su factorial. Se interrumpe cuando se introduce un número negativo. Su finalidad es indicar cómo pasan los parámetros al subprograma y cómo son devueltos por él mismo en Basic. En los ejercicios propuestos al final del capítulo se presentan programas con más peso específico a propósito para utilizar el mismo subprograma factorial.

El algoritmo en que se basa el presente programa es:

Repetir el proceso:

> Introducir un número.
> Transferirlo al subprograma factorial.
> Presentar en salida el factorial entregado por el subprograma.

Hasta que se introduzca un número negativo.

Variables

Para mantener la idea de variables independientes en módulos independientes, el programa no utiliza las variables **N** y **F** salvo como parámetros, y no utiliza para nada la variable **K**. Por tanto, las variables son:

I Número introducido.
J Factorial.

Diagrama de flujo

La figura 9.4 ilustra el flujo de control de este programa.

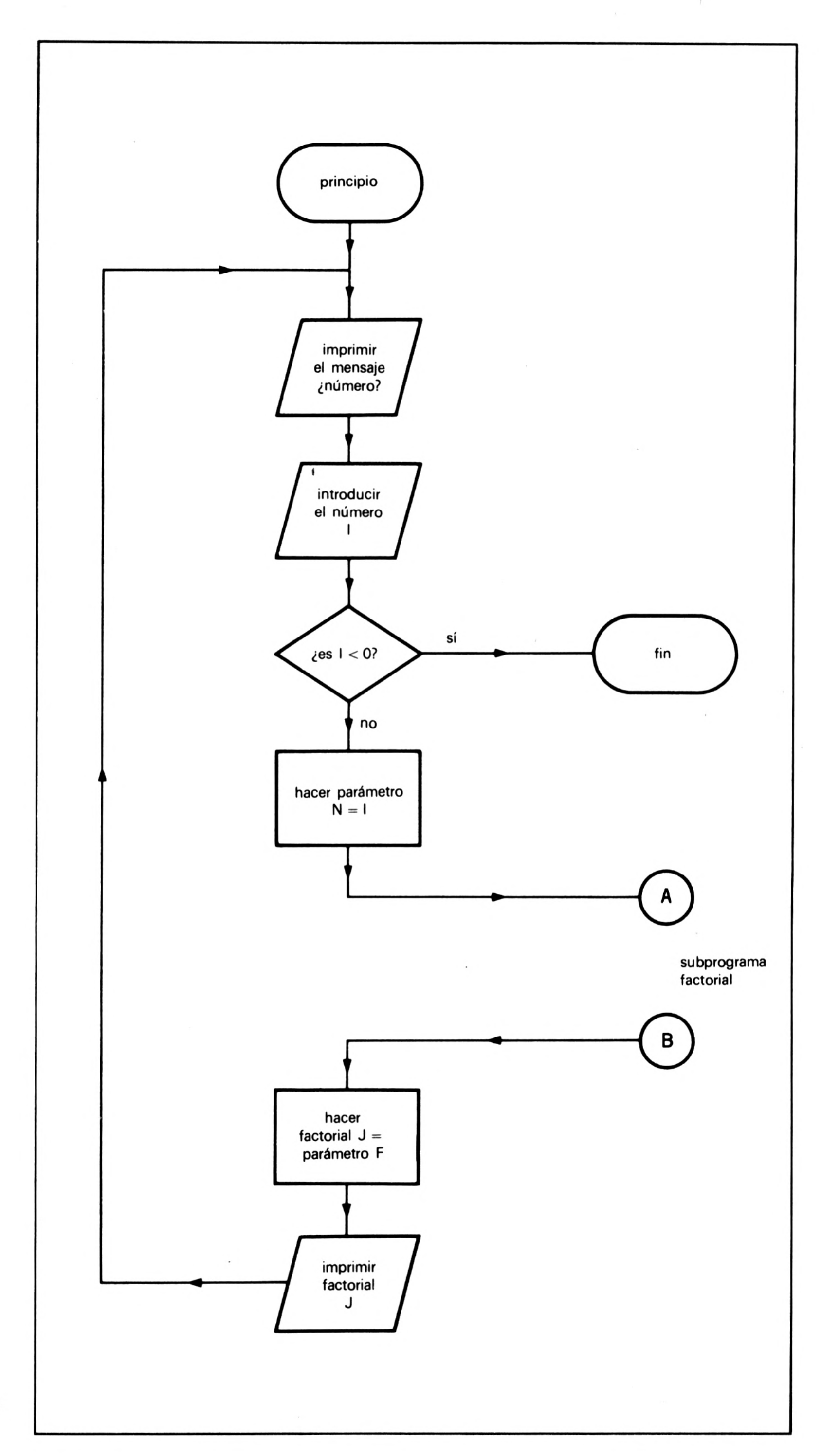


Figura 9.4
Diagrama de flujo del programa principal del ejemplo 9.2

Programa

```
100 REM PROGRAMA EJEMPLO 9.2
105 REM PROGRAMA PRINCIPAL DE DEMOSTRACION
110 REM PARA EL SUBPROGRAMA FACTORIAL
115 REM
200 PRINT "NUMERO?" ;
205 INPUT I
210 IF I < 0 THEN STOP
215 LET N = I
220 GOSUB 1000 : REM SUBPROGRAMA FACTORIAL
225 LET J = F
230 PRINT "FACTORIAL" ; J
235 GOTO 200
240 REM FINAL DEL PROGRAMA PRINCIPAL
```

Puntos de interés

- Las líneas 215 y 225 indican de qué forma se asignan y leen los parámetros antes y después de llamar al subprograma.
- El programa no es la ejecución práctica estricta del algoritmo "repetir... hasta que", porque la condición para terminar la repetición (línea 210) se encuentra en el centro del bucle.
- En la línea 210 se emplea la versión perfeccionada de la instrucción **IF**.

9.6 Conclusión

En este capítulo se ha presentado la noción de subprograma y se ha ilustrado la realización práctica de la misma en Basic. Los puntos que merecen más atención son:

- Un subprograma es una porción de un programa que lleva a cabo una tarea específica, sirviéndose para ello de ciertos datos. El subprograma cumple el papel de un "ladrillo" dentro de la estructura general del programa.
- El subprograma se llama desde el programa principal. Una vez terminado, el control vuelve al punto de llamada de dicho programa principal. También puede llamarse a un subprograma desde otro.
- Aunque en Basic el programa principal y los subprogramas comparten las variables, es aconsejable utilizar variables dife-

rentes en uno y en otros y hacer las oportunas transferencias de datos por medio de parámetros.

- Se llama global a cualquier variable que se emplea en todo el programa, incluyendo a los subprogramas. Las variables locales sólo se usan dentro de un único subprograma.

Para terminar, sólo decir que muchas de las ideas presentadas en este capítulo están íntimamente relacionadas con el concepto de diseño de programa que se discutirá en el 11.

9

Ejercicio

1. Defina brevemente los siguientes términos: subprograma, programa principal, llamada, retorno, recurrencia, variable local, parámetro, variable global.
2. Indique algunas de las carencias que caracterizan la ejecución de subprogramas en Basic.
3. *a*) Modifique el programa ejemplo 9.1 para incluir en el mismo cálculos sencillos de porcentajes de la forma siguiente:

   Añada la operación cálculo del porcentaje, de símbolo % y código 6, que lleva a cabo el cálculo

   $$\mathbf{A} = \frac{\mathbf{A} \times \mathbf{B}}{100}$$

   Por ejemplo: si se introducen los números 60 y 240, se lleva a cabo el cálculo del 60 por 100 de 240 (= 144).

   b) La inclusión en la calculadora de la operación de borrado obliga a introducir correcciones en el programa principal y en el subprograma de verificación y a añadir un nuevo subprograma.

   La operación de borrado, de símbolo C y código 7, pone los dos registros a 0 y el código **P** de la operación anterior a 1 (copiar). Además, se ejecuta en el mismo momento en que se introduce, sin esperar a que se convierta en la operación anterior.

   Escriba un subprograma que realice la operación de borrado y modifique el programa principal y el subprograma de verificación para acoplarlo al conjunto.
4. La función factorial sirve, entre otras cosas, para calcular las **probabilidades binomiales**, como ilustra este ejemplo:

   Un empleado coge todos los días un determinado tren para ir a trabajar, pero sabe que existe cierta probabilidad de que ese tren sea anulado. Teniendo en cuenta esa probabilidad, ¿cuáles son las de que en una semana sean cancelados 0, 1, 2 y hasta 5 trenes?

   Suponiendo que la cancelación del tren en un día cualquiera es independiente de lo que haya ocurrido en días anteriores, la fórmula que determina las probabilidades pedidas es:

   $$c = \frac{n!}{(n - r!)r!} \times p^r(1 - p)^{n-r}$$

Donde *p* es la probabilidad de que se cancele un tren, *r* el número de cancelaciones en una semana (0, 1, 2, ..., 5), *n* es el número de jornadas laborales de la semana (5 en este caso) y *c* es la probabilidad de que se produzcan *r* cancelaciones.

Escriba un programa que acepte la introducción de la probabilidad *p*, verifique si está en el intervalo 0-1 y calcule el valor de *c* para los valores de *r* comprendidos entre 0 y 5 (es decir, las probabilidades de que se cancelen 0, 1, 2... hasta 5 trenes). Utilice el subprograma factorial del programa ejemplo 9.2.

5. Vuelva a escribir el programa ejemplo 7.2 (la aplicación de manipulación de matrices) en forma de un programa principal y un conjunto de subprogramas. En el programa principal se encuentran los módulos de inicialización y control, y cada módulo de operación se transforma en un subprograma independiente.

El módulo de control debe rehacerse por completo, para lo que puede seguirse un método similar al del programa ejemplo 9.1. Por el contrario, bastan unas pocas modificaciones para transformar en subprogramas los módulos de operaciones.

6. Escriba un paquete estadístico con módulos para ejecutar algunas de las operaciones indicadas a continuación o todas ellas:

Introducir un conjunto de números en una matriz.
Calcular y mostrar la media de dichos números.
Calcular y mostrar la desviación típica.
Calcular y mostrar el intervalo que contiene a los números (máximo-mínimo).
Mostrar el conjunto de los números.

El módulo de control es el programa principal, que acepta órdenes del usuario y transfiere el control a los subprogramas según sea necesario. En un programa de este tipo es aconsejable que todos los módulos trabajen con la misma matriz de números que, por tanto, debe ser una variable global. En Basic, el empleo de una matriz como parámetro es causa de problemas.

7. Escriba un subprograma que acepte como parámetro una variable que represente una serie de caracteres alfanuméricos y devuelva como salida otra variable con la misma serie de caracteres, pero habiendo sustituido por mayúsculas todas las minúsculas.

Utilice ese subprograma en un programa principal que acepte la introducción de un texto compuesto por varias palabras, identifique la primera letra de cada una de ellas y la sustituya por su equivalente mayúscula. Utilice nombres de variables diferentes en el subprograma y en el programa principal.

8. Vuelva a escribir el programa ejemplo 6.2 sustituyendo los segmentos que empiezan en las líneas 6200 y 6300 por un solo subprograma y haciendo las oportunas transferencias de parámetros entre dicho subprograma y el programa principal.

9. Escriba un programa que lea una tabla de información dispuesta en una matriz de dos dimensiones y a continuación permita al usuario realizar una o más de las siguientes operaciones:

Localizar un elemento de la tabla e imprimir o presentar en pantalla la fila de la misma que lo contiene.
Sumar o hallar la media de cualquier columna de la tabla.
Producir una columna adicional que contenga las sumas o las diferencias de las cifras correspondientes de otras columnas.

La tabla puede contener información geográfica, sociológica o química y puede leerse por medio de instrucciones **DATA** contenidas en el programa.
Escriba un programa que simule el comportamiento de un microprocesador encargado de controlar una máquina en una fábrica. La máquina puede operar en varias modalidades, consistentes cada una de ellas en un ciclo de señales procedentes del microprocesador. Las instrucciones que recibe éste se refieren a la modalidad de funcionamiento y al número de ciclos de instrucciones que deben ejecutarse en dicha modalidad. Un ciclo de instrucciones de 12 segundos podría ser el siguiente:

Tiempo cero,	señal	en	línea C
tiempo 3,	señal	en	línea A
tiempo 4,5,	señal	en	línea C
tiempo 5,	señal	en	línea B
tiempo 8,	señal	en	línea D
tiempo 10,5,	señal	en	línea C.

Simule las señales de control de la máquina mediante caracteres formados en pantalla.


RESET
RETURN

10 Manipulación de ficheros

Los capítulos vistos hasta ahora se han referido a la entrada, el proceso y la salida de datos. Este va dedicado al estudio de la transferencia de datos hacia y desde los soportes de almacenamiento. Dado que esos datos se organizan en forma de ficheros, su manejo se conoce con el nombre de **manipulación de ficheros.**

El capítulo empieza con una discusión general sobre la naturaleza y las aplicaciones de los ficheros, para examinar a continuación su manipulación en Basic y una serie de técnicas de tratamiento de los mismos.

Por desgracia, la manipulación de ficheros es uno de los aspectos del Basic que más directamente depende del ordenador con que se trabaje. (Aunque casi toda la discusión se conduce en términos generales, los programas ejemplos se realizan en la versión del Basic del IBM PC.) Al final del capítulo se esbozan las equivalencias con otras versiones también muy conocidas.

Justamente a causa de su dependencia del ordenador en uso, la manipulación de archivos es una técnica que no se utiliza mucho en el resto de este libro y, por la misma razón, este capítulo puede saltarse sin menoscabo de la continuidad.

10.1

Naturaleza y aplicaciones de los ficheros

En su sentido más amplio, un archivo es una colección organizada de información. Por lo que respecta a este curso de programación, un archivo es una colección estructurada de datos almacenados en la memoria auxiliar del ordenador, generalmente sobre un soporte magnético en forma de disco. Un archivo posee ciertas propiedades, que estudiaremos en esta sección.

Un archivo queda identificado por un **nombre.** En un mismo sistema, no puede haber dos archivos con el mismo nombre. Un archivo tiene también una **estructura** rígidamente definida. En la práctica, los ficheros pueden tener diferentes estructuras de diverso grado de complejidad, aunque los considerados aquí serán de estructura sencilla: un conjunto de **registros** formados a su vez por una serie de **campos,** en cada uno de los cuales se almacena un dato. Todos los registros de un fichero tienen el mismo número de campos.

Los datos pueden **leerse** de un fichero o **escribirse** en el mismo. Cuando un fichero está listo para aceptar la entrada de datos, se dice que está **abierto para escritura.** De la misma forma, se dice que está **abierto para lectura** cuando los datos que contiene pueden leerse. Una limitación importante a tener en cuenta es que un fichero nunca puede estar simultáneamente abierto para lectura y escritura.

10.2

Manipulación de ficheros en Basic

Casi todas las versiones del Basic cuentan con recursos de manipulación de ficheros, aunque, como ya se ha dicho más arriba, las instrucciones específicas dependen del tipo de ordenador. Sin embargo, esas operaciones son básicamente iguales en todos los casos, y se discuten en términos generales en esta misma sección. Más adelante veremos su ejecución en una versión particular del lenguaje.

Las operaciones realizables sobre un fichero son las siguientes:

- Un fichero puede **crearse.** En este proceso, el fichero recibe un nombre, habitualmente una serie de caracteres alfanuméricos de cualquier longitud, y se abre para escritura.
- En un fichero pueden **escribirse** datos; la operación se realiza registro a registro, trabajando siempre en orden a partir del primero de ellos.

- Cuando se termina la escritura, el fichero se **cierra.** Hecho esto, ya no pueden escribirse más registros.
- Un fichero puede abrirse para lectura, dando su nombre para identificarlo.
- Los datos de un fichero pueden **leerse,** operación que se ejecuta registro a registro, trabajando en orden desde el primero de ellos.
- El nombre de un fichero puede **cambiarse** mediante una operación de cambio de nombre.
- Si un fichero deja de hacer falta, puede **borrarse.**

En los ordenadores controlados por un microprocesador hay que contar con una limitación adicional en virtud de la cual no es posible leer y escribir un fichero simultáneamente, lo que a veces obliga a crear ficheros provisionales para almacenar datos a lo largo del proceso.

10.3 Creación de un fichero

En el proceso de creación de un fichero, éste recibe un nombre y queda listo para el almacenamiento de datos. Primero se hace una verificación que tiene por objeto comprobar si el nombre existe ya en el disco que sirve de soporte al fichero. En tal caso, el usuario puede elegir entre eliminar el archivo anterior que ya tenía ese nombre o elegir otro para el nuevo.

En la versión del Basic escogida (IBM PC), las instrucciones para crear un fichero incluyen la palabra OPEN, el nombre del fichero, las palabras FOR OUTPUT o FOR APPEND, para crearlo nuevo o añadir datos a uno ya existente, respectivamente, y el número de canal. Por ejemplo:

```
OPEN "REGPAGOS" FOR APPEND AS #10
```

abre un fichero llamado REGPAGOS para añadirle al final lo que queramos, mientras que con la sentencia:

```
OPEN "REGPAGOS" FOR OUTPUT AS #10
```

abrimos el fichero REGPAGOS para escribir, borrando los datos existentes anteriormente.

10.4

Escritura en un fichero

Un fichero se escribe registro a registro. Para ello se usa una versión modificada de la instrucción **WRITE**; cada una de esas instrucciones crea un registro.

En la versión del Basic elegida, a la instrucción **WRITE** sigue el número de canal (también **# 10**) y una lista de variables o constantes que comprenden un registro del fichero. Por ejemplo:

```
WRITE #10, A, B$, C, D$
```

escribe un registro del fichero con los valores de las variables **A, B$, C** y **D$**.

10.5

Cierre de un fichero

Una vez escritos todos los datos, el fichero se cierra. Mediante esta operación se inserta un indicador de **fin de fichero,** que completa el almacenamiento de los datos. Si tras escribir los datos el fichero no se cierra, aquéllos se pierden. En la versión del Basic elegida, el archivo que se está escribiendo se cierra mediante la instrucción

```
CLOSE #10
```

El indicador de fin de fichero mencionado puede emplearse en algunas operaciones de programación, pero no en todas las versiones del Basic. Es aconsejable terminar siempre los ficheros con un registro de fin de fichero especialmente creado con ese objeto.

10.6

Programa ejemplo 10.1

La finalidad de este programa es crear un archivo que contenga el nombre de una persona, su dirección y su número de teléfono. El programa se refiere sólo a la creación de esta agenda, pero a lo largo del capítulo se presentarán otros destinados a acceder a ella y mantenerla al día.

Método

La primera cuestión a resolver es la estructura del fichero. Una elección obvia sería la de tres campos alfanuméricos por registro, uno para el nombre de la persona, otro para la dirección y otro para el número de teléfono. (No es aconsejable reservar para éste un campo numérico, porque habrá que archivar números grandes escritos en la forma habitual.)

Un registro fin de fichero adecuado sería

```
FINOMBRE  FINDIRECCION  00000
```

El algoritmo del programa es:

Crear el fichero agenda.
Repetir el proceso.

  Introducir un registro (nombre, dirección, teléfono).
  Escribir el registro en el fichero.

Hasta llegar al registro fin de fichero.

Variables

N$ Nombre.
D$ Dirección.
T$ Número de teléfono.

Diagrama de flujo

Se ilustra en la figura 10.1.

Programa

```
100 REM PROGRAMA EJEMPLO 10.1
105 REM CREACION DE UNA GUIA DE NOMBRES, DIREC
        CIONES Y NUMEROS DE TELEFONO
110 REM
200 OPEN "NOMDITELGUIA" FOR APPEND AS #10
205 PRINT "CREACION DE UNA GUIA DE NOMBRES, DI
           RECCIONES Y NUMEROS DE TELEFONO"
210 PRINT
215 PRINT "INTRODUZCA UN NOMBRE, LA DIRECCION Y
           EL NUMERO DE TELEFONO"
220 PRINT "A LA VEZ"
225 PRINT
230 PRINT "FINALICE LA ENTRADA CON 'FINOMBRE',
           'FINDIRECCION', 0000"
235 PRINT
```

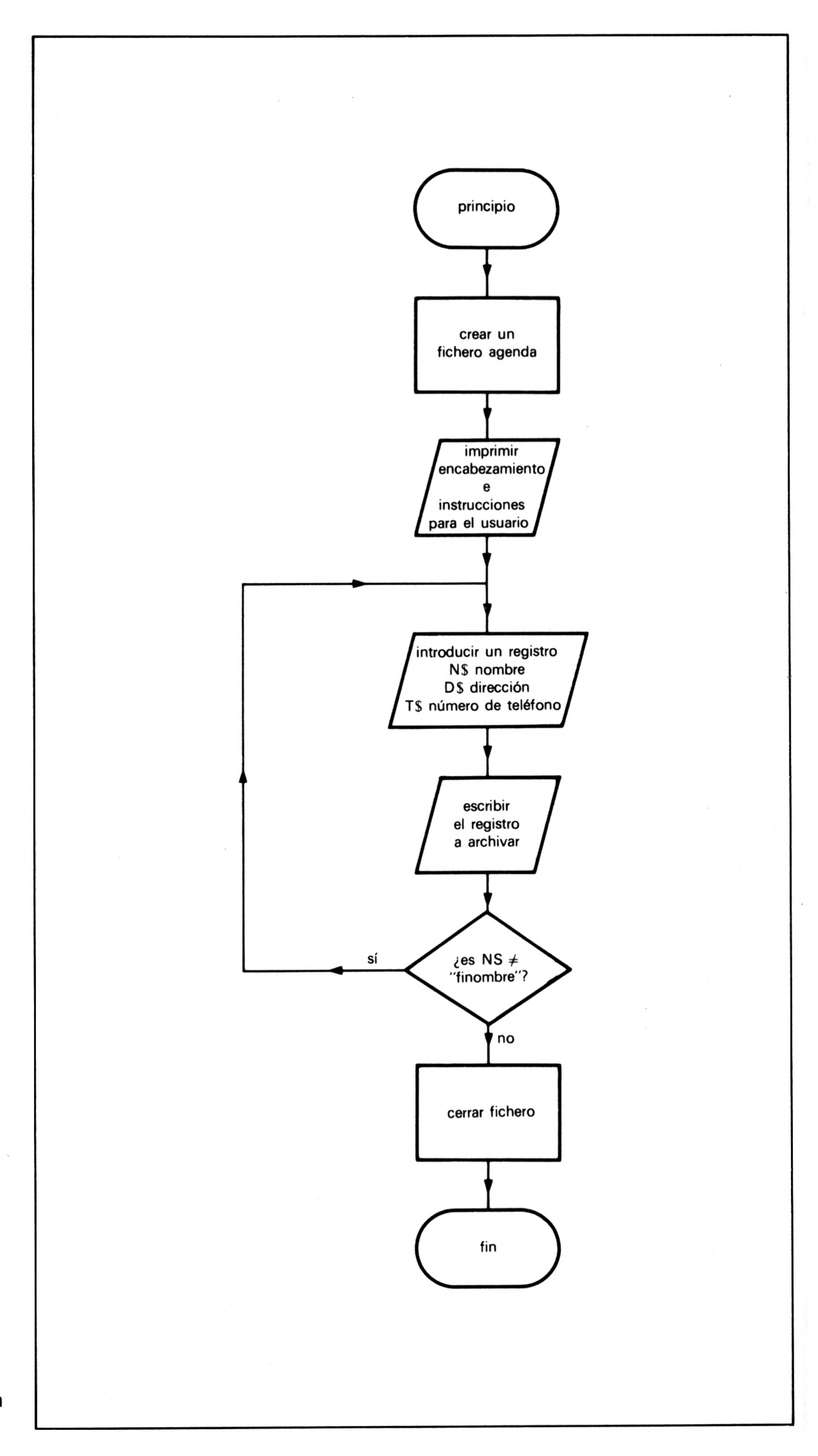


Figura 10.1
Diagrama de flujo del programa ejemplo 10.1

```
240 REM
300 INPUT N$, D$, T$
305 WRITE #10, N$, D$, T$
310 IF N$ <> "FINOMBRE" THEN 300
315 REM
400 CLOSE #10
405 END
```

Puntos de interés

- El registro fin de fichero se usa también como condición de fin de entrada.
- Una vez terminada la escritura del fichero, éste se cierra.

10.7 Apertura de un fichero para lectura

Cuando se abre un fichero, éste queda listo para lectura. Se empieza por efectuar una verificación para comprobar si el fichero que ha de abrirse existe y si se encuentra abierto para escritura. Si el fichero no se encuentra o si está abierto para escritura, aparece un mensaje de error y el usuario debe realizar las correcciones necesarias.

En la versión del Basic elegida, la instrucción de apertura de un fichero para lectura incluye la palabra **OPEN FOR INPUT**, el nombre del fichero y el número de canal (también **AS # 10**). Por ejemplo:

```
OPEN "REGPAGOS" FOR INPUT AS #10
```

abre para lectura el fichero **REGPAGOS**.

10.8 Lectura de un fichero

Un fichero se lee registro a registro, empezando por el primero de ellos y avanzando en el orden en que fueron escritos. No puede saltarse ninguna de las posiciones del fichero.

En la versión del Basic elegida se usa para leer archivos una instrucción **INPUT** modificada. En ésta, a la palabra **INPUT** sigue el número de canal (también **# 10**) y una lista de variables. Dentro de un registro, cada campo debe tener un nombre de variable. En otras palabras, las variables utilizadas en la instrucción **INPUT # 10** deben

ser iguales a las usadas en la **WRITE # 10** para escribir ese campo del registro. Por ejemplo:

```
INPUT #10, J, K$, L, M$
```

leerá un registro del fichero escrito en el ejemplo de la sección 10.4.

Hay que tener cuidado para no tratar de leer más allá del fin de un fichero. Aunque algunos sistemas pueden utilizar el indicador de fin de fichero para determinar el momento en que se ha terminado éste, es aconsejable verificar el registro de fin de fichero insertado durante la operación de escritura (véase sección 10.5).

10.9

Cambio de nombre de un fichero

El nombre de un fichero puede modificarse por medio de la orden **NAME.** Primero se comprueba si el fichero cuyo nombre quiere cambiarse existe y si el nombre nuevo que desea utilizarse está ya en uso. Si se cumple la primera condición, se presenta un mensaje de error para que el usuario lleve a cabo las correcciones necesarias. Si el nombre existe, el usuario puede elegir entre anular el archivo que posee el nombre escogido y cambiar éste por otro.

La instrucción **NAME** incluye el nombre nuevo y el antiguo, en este orden:

```
NAME "FICHPAGOS" AS "REGPAGOS"
```

cambia el nombre del fichero **FICHPAGOS** por el **REGPAGOS**. También pueden usarse variables como nombres de fichero.

La instrucción **NAME** es particularmente útil para poner al día los ficheros. Para ello, se leen los registros de la versión en curso del fichero, se procesan y se escriben en un fichero provisional, al que a continuación se da el nombre de la versión actualizada del de partida.

10.10

Borrado de un fichero

La instrucción **KILL** se emplea para eliminar un fichero cuando ya no es necesario. En la versión del Basic elegida aquí, a la palabra **KILL** sigue el nombre del fichero. Por ejemplo:

```
KILL "REGPAGOS"
```

Si el nombre del fichero no se localiza, no se genera ningún mensaje de error.

10.11

Programa ejemplo 10.2

Este programa utiliza los nombres, direcciones y números de teléfono de la agenda creada en el programa ejemplo 10.1. El usuario puede servirse de él para buscar en la agenda direcciones y teléfonos.

Método

Se introduce el nombre sobre el que se desea información y el fichero agenda se abre para lectura. Se utiliza un bucle para recorrerlo hasta encontrar un registro idéntico al nombre introducido. El método puede resumirse como sigue:

Repetir el proceso:

Leer un registro.
Si el nombre coincide con el introducido, mostrar la dirección y el teléfono.

Hasta localizar el registro deseado o hasta llegar al final del fichero.

Observe que hay dos condiciones de interrupción del bucle.

Variables

I$ Nombre introducido.
N$ Nombre (del registro).
D$ Dirección (del registro).
T$ Número de teléfono (del registro).

Diagrama de flujo

El flujo de control de este programa se ilustra en la figura 10.2.

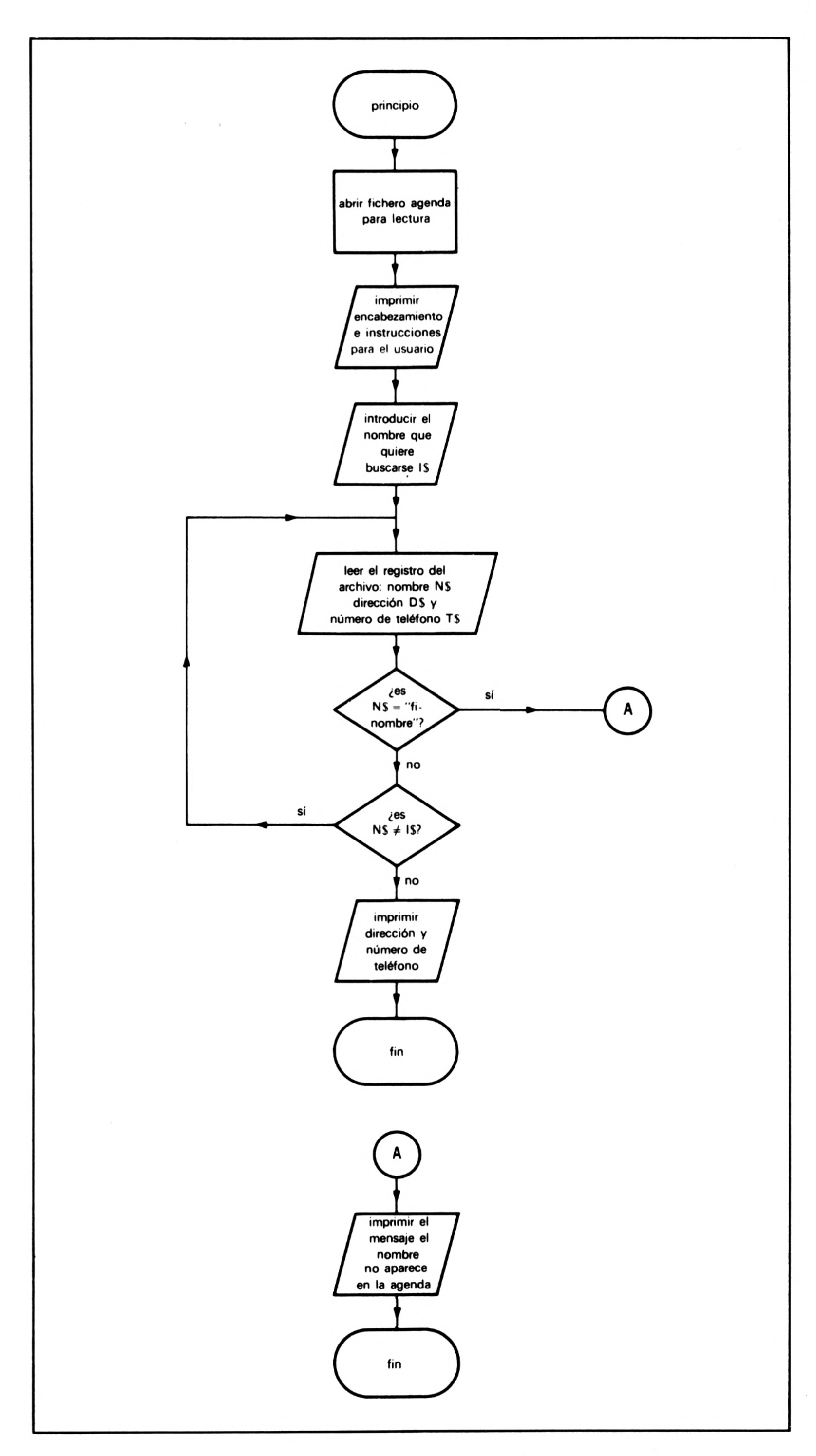


Figura 10.2
Diagrama de flujo del programa ejemplo 10.2

Programa

```
100 REM PROGRAMA EJEMPLO 10.2
105 REM ACCESO A UNA GUIA DE NOMBRES, DIRECCIO
        NES Y NUMEROS DE TELEFONO
110 REM
200 OPEN "NOMDITELGUIA" FOR INPUT AS #10
205 PRINT "ACCESO A UNA GUIA DE NOMBRES, DIREC
          CIONES Y NUMEROS DE TELEFONO"
210 PRINT
215 PRINT "INTRODUZCA EL NOMBRE QUE SE VA A BUS
          CAR EN"
220 PRINT "LA GUIA"
225 INPUT I$
230 REM
300 INPUT #10, N$, D$, T$
305 IF N$ = "FINOMBRE" THEN 330
310 IF N$ <> I$ THEN 300
315 PRINT D$
320 PRINT T$
325 STOP
330 PRINT "EL NOMBRE NO SE ENCUENTRA EN LA GUIA"
335 END
```

Puntos de interés

- Observe que el algoritmo de repetición se interpreta en las líneas 300 a 310 del programa.
- Las variables de la línea 300 deben coincidir en número y tipo con las usadas en la instrucción **WRITE** usada para escribir el fichero (línea 300 del programa ejemplo 10.1). No es preciso que los nombres de las variables sean idénticos.
- Este programa sólo será capaz de localizar un nombre si se introduce exactamente de la misma forma en que está archivado. Es, pues, preciso sistematizar la escritura de nombres, apellidos e iniciales.
- La técnica empleada en este ejemplo se llama **búsqueda secuencial,** y se discutirá en términos generales en las secciones 16.2 y 16.3.

10.12 Instrucciones de manipulación de ficheros utilizadas en otras versiones del Basic

Como ya se ha dicho, la manipulación de ficheros es uno de los aspectos del lenguaje Basic más dependientes del ordenador.

Por ello, ha sido necesario elegir una versión del mismo para discutir detalladamente los ejemplos presentados en este capítulo.

Para que los usuarios de ordenadores de otras marcas puedan adaptar el material expuesto a sus necesidades, se han recogido en la tabla de la figura 10.3 las instrucciones encargadas de ejecutar las operaciones descritas en la sección 10.2 en las diferentes versiones del Basic más conocidas.

Por lo general, bastará con cambiar unas instrucciones por otras, sin necesidad de modificar la estructura lógica del programa.

Figura 10.3
Instrucciones de manipulación de ficheros en diversas versiones del Basic

Operación (E$ es el nombre del fichero; A y B$ son variables)	IBM PC (# 1 es el número de canal)	Commodore Pet Versión del Basic 4,0	Tandy TRS 80 Nivel II/Basic para sistema de discos (B es el número de memoria tampón)	Basic Apple 2 (A D$ se asigna el valor de CTRL-D)	ICL 2903 Basic sistema educativo	Basic BBC Microcomputer (N es el número de canal)
Crear un fichero y abrirlo para escritura	OPEN F$ AS # 1	DOPEN # 1,F$,W	OPEN "O", B, F$	PRINT D$; "OPEN FNAME" PRINT D$; "WRITE FNAME"	FILE # N: F$	N = OPENOUT # ("FNAME")
Escribir un registro de un fichero	PRINT # 1, A	PRINT #1, A, B$	PRINT # B, A, B$	PRINT A, B$	PRINT # N: A, B$ OR WRITE # N: A, B$	PRINT # N, A, B$
Cerrar un fichero	CLOSE # 1	DCLOSE # 1	CLOSE B	PRINT D$; "CLOSE FNAME	FILE # N:"#	CLOSE # N
Abrir un fichero para lectura	OPEN F$ AS # 1	DOPEN # 1, F$	OPEN "I", B,F$	PRINT D$; "OPEN FNAME" PRINT D$; "READ FNAME"	FILE # N: F$	N = OPENIN # ("FNAME")
Leer un registro de un fichero	INPUT #1, A	INPUT # 1, F$	GET B INPUT # B, A, B$	INPUT A, B$	INPUT # N: A, B$ OR READ # N: A, B$	INPUT # N, A, B$
Cambiar el nombre de un fichero	RENAME G$, F$	RENAME F$ to G$	—	RENAME OLDNAME, NEWNAME	—	—
Borrar un fichero	ERASE F$	SCRATCH F$	KILL F$	DELETE FNAME	SCRATCH # N	

10.13
Conclusión

Este capítulo ha sido una breve introducción a las operaciones de transferencia de datos a y desde la memoria auxiliar. Se han presentado varias técnicas de manipulación de ficheros y se han aportado algunos ejemplos que llevan a la práctica esas técnicas en una versión determinada del Basic.

Los puntos más importantes, independientes todos ellos de la versión del Basic usada, son, para escribir datos en un fichero:

- Un fichero se crea y se abre para escritura.
- Los datos se escriben en el mismo, registro a registro.
- Una vez escritos todos los registros, el archivo se cierra.

Los pasos para leer datos de un fichero son:

- El fichero se abre para lectura.
- Los datos del fichero se leen registro a registro, en el mismo orden en que fueron escritos.

Ejercicio

1. Defina brevemente los siguientes términos: fichero, registro, campo, abierto para escritura, abierto para lectura, cierre de un fichero, indicador de fin de fichero.
2. Modifique el programa ejemplo 10.2 de forma que permita buscar varios nombres seguidos en la agenda.
3. Al **actualizar** un fichero se eliminan registros ya innecesarios, se modifican otros y se insertan algunos nuevos.

   Como en la mayor parte de las versiones del Basic es imposible leer y escribir un fichero al mismo tiempo, la técnica usual de actualización consiste en leer la versión actual del fichero, realizar las correcciones y escribir los registros actualizados en un fichero provisional, que a continuación recibe el nombre del fichero original actualizado; la versión original se borra o se archiva aparte.

   Escriba un programa de actualización de la agenda empleada en los ejemplos 10.1 y 10.2. Se sugiere el procedimiento siguiente:

   Introducir en una matriz los nombres de todos los registros que han de eliminarse o corregirse.
   Repetir el proceso:

   Leer un registro del fichero en curso.
   Si el nombre no coincide con ninguno de los de la matriz, copiar el registro en un fichero provisional.

   Hasta llegar al registro final del fichero.
   Repetir el proceso:

   Introducir los datos para un nuevo registro o un registro ya corregido.
   Escribir este registro en el fichero provisional.

   Hasta llegar al final de la entrada.
   Escribir un registro de fin de fichero en el fichero provisional.
   Cambiar el nombre del fichero provisional por el del fichero actualizado y borrar la versión original.

4. Escribir un conjunto de programas para crear un sistema de almacenamiento y recuperación de documentos con arreglo a las siguientes especificaciones:

   El **programa de entrada** crea un fichero para almacenar un documento y permite al usuario darle un nombre. A continuación se introduce el texto del documento línea por línea. Cada línea queda archivada en un registro.

   El **programa de actualización** da acceso a un documento de un fichero y lo almacena línea por línea en una matriz, de manera que a cada elemento de ésta corresponde una línea de aquél. Hecho esto, las líneas pueden corregirse, anularse o insertarse. El documento vuelve a escribirse en una nueva versión del fichero.

   El **programa de salida** da acceso a un documento de un fichero y lo imprime.
5. Algunos de los programas realizados con anterioridad pueden ampliarse mediante la inclusión de módulos capaces de escribir o leer datos en un fichero, de manera que los datos puedan conservarse una vez

ejecutado el programa. Entre esos programas están el paquete de tratamiento de matrices, ejemplo 7.2, el diccionario de abreviaturas, ejemplo 7.1, y el de tratamiento de vectores, pregunta 5 del ejercicio 7.

6. Escribir un conjunto de programas para crear, actualizar y acceder a un fichero de datos de los alumnos de una escuela o los miembros de un club juvenil. Los programas solicitados podrían tener las siguientes características:

   Un programa de **entrada** que permita incorporar datos por medio de un teclado, verificarlos y editarlos en caso necesario, y almacenarlos en un fichero con nombre.

   Un programa de **actualización** que permita actualizar o borrar registros de cualquier fichero con nombre o añadir al mismo registros nuevos.

   Un programa de **acceso** que permita examinar en pantalla o imprimir la información contenida en los registros seleccionados.

7. El servicio Videotex da a los usuarios acceso a la información contenida en un ordenador central por medio del teléfono, y presenta dicha información en la pantalla de un televisor. El contenido de la pantalla corresponde a una "página" de información. El usuario controla el sistema por medio de un sencillo teclado. Todas las "páginas" tienen al pie información para acceder a otras por medio de las teclas adecuadas. La estructura general de las "páginas" es un árbol (véase capítulo 21).

   Elija un motivo cualquiera y escriba una serie de "páginas" de información sobre el mismo. Haga coincidir cada una de esas páginas con el tamaño de la pantalla de su ordenador, e incluya en todas ellas instrucciones para acceder a las anteriores, a la estructura del árbol y a las siguientes. Diseñe también un esquema para nombrar las "páginas" de acuerdo con su posición en la estructura de árbol.

   Escriba un programa para crear una serie de ficheros de datos capaces cada uno de almacenar una "página" y nombrados de acuerdo con el esquema anterior. El fichero debe incluir también una lista con los caracteres necesarios para transferir el control a otras páginas. Redacte igualmente un programa de verificación y, si fuese necesario, edite el contenido de una "página".

   Redacte a continuación un programa similar al Videotex para presentar la "página" "superior" del árbol y que ofrezca al usuario la posibilidad de recorrer las demás "páginas", siguiendo para ello las instrucciones de acceso dispuestas al pie de cada una de ellas.

   Pueden someterse a este tratamiento temas como un archivo de noticias o de información deportiva, un recetario de cocina, unos apuntes sobre alguna materia de estudio, una guía local de espectáculos, un libro de instrucciones de reparación de una bicicleta o una motocicleta o una guía de viaje.


RESET
RETURN

11
Diseño de programas

En los capítulos anteriores se han presentado una serie de elementos del Basic junto a diversos programas pensados para ilustrar el uso de dichos elementos y que, por ello, se han simplificado y acortado deliberadamente. Pero, en la práctica, los programas no suelen adaptarse a las características del lenguaje tan limpiamente como en esos ejemplos. Los programas reales, como los puentes o las casas, tienen que satisfacer una serie de exigencias muy rigurosas.

En este capítulo nos enfrentamos al problema de **diseñar** programas. Discutiremos los objetivos que persigue el correcto diseño de programas y presentaremos algunas técnicas de diseño, de las que una se discute con cierto detalle y se aplica a dos ejemplos.

El material de este capítulo es independiente de cualquier lenguaje de programación particular, y se refiere al proceso por medio del cual se transforma la especificación de una tarea o un problema en un programa. Las técnicas presentadas se emplearán continuamente en el resto del libro, y son imprescindibles para cualquiera que desee escribir programas dignos de tal nombre, aunque sólo llegan a dominarse con paciencia y con la práctica.

11.1

¿Qué es el diseño de programas?

El diseño de programas es una forma de abordar la labor de programación. Es una técnica, pero también una actitud mental. Aporta la idea de dirección al difícil proceso de desarrollar un programa y garantiza que el resultado final tendrá un nivel de calidad aceptablemente alto.

Con demasiada frecuencia, se encuentran programas que están manifiestamente "hechos sobre la marcha". Semejantes programas raramente son correctos, y por lo general resultan incomprensibles. La única manera de evitar estos problemas es planificar meticulosamente los programas antes de escribirlos. En este capítulo examinaremos precisamente algunos métodos de planificación.

11.2

Objetivos de un programa bien diseñado

Antes de pasar a los métodos de diseño de programas, conviene reflexionar sobre lo que se pretende conseguir. Un programa bien diseñado debe satisfacer una serie de objetivos, que deben tenerse en cuenta a lo largo de todo el proceso de diseño. La calidad del resultado puede evaluarse por la medida en que se cumplen dichos objetivos.

En los próximos párrafos examinaremos esos objetivos y discutiremos algunas de las limitaciones que suelen presentarse en la práctica.

Corrección

Un programa no vale de nada si no funciona bien. No basta con que produzca un resultado correcto sólo cuando se le proporciona una entrada correcta: debe funcionar correctamente bajo todas las condiciones imaginables, aunque ésta es una exigencia dificilísima de satisfacer en la práctica.

En la mayor parte de los casos, la corrección depende de la atención dedicada a los algoritmos del proceso y de la realización de una serie de pruebas exhaustivas. En algunos casos, la corrección de un programa se verifica de forma similar a la demostración de ciertos teoremas matemáticos. No obstante, las técnicas de comprobación de la corrección de los programas quedan fuera del alncace de este curso.

Estructura clara

Se llama estructura de un programa a la forma en que sus diversas partes se unen para formar el todo. La estructura de un programa debe reflejar la de los datos que procesa y también el flujo lógico de las operaciones que realiza.

La forma de estructuración más aceptada es la llamada **programación modular** (o **programación estructurada**), siendo un módulo una parte del programa que realiza una operación específica. La entrada y la salida de cada módulo están definidas con toda precisión, lo mismo que el proceso realizado en su interior. Sin embargo, los detalles relativos al trabajo del proceso no suelen especificarse.

Un programa se estructura descomponiendo la tarea que realiza en una serie de procesos, cada uno de los cuales se ejecuta en un módulo. A continuación se construye el programa conectando los diversos módulos. Los datos que entran y salen de cada módulo conforman el **interfaz** entre éste y el resto del programa.

Un módulo no debe provocar **efectos secundarios,** lo que significa que debe limitarse a realizar la tarea que le ha sido encomendada sin afectar a los datos que no debe procesar.

Facilidad de modificación

Es muy poco probable que la tarea que debe ejecutar un programa se mantenga mucho tiempo sin variaciones, por lo que casi todos los programas deben modificarse de vez en cuando. Precisamente lo que revela la calidad de un programa es la facilidad que tiene para aceptar modificaciones —pequeñas o grandes— sin resentirse en su estructura general. Si un programa es modular, la mayor parte de los cambios se realizan corrigiendo los módulos afectados o cambiándolos por otros, de manera que la estructura general permanece invariable.

Facilidad de lectura

Aunque los programas suelen ir acompañados de documentación que explica su funcionamiento, el propio texto de un programa bien escrito debe resultar claramente legible. Un programa legible es también fácil de entender, lo que simplifica la corrección de errores y la realización de modificaciones. La legibilidad queda garantizada por la elección acertada de los nombres de las variables, por la disposición clara de las instrucciones y por el uso de comentarios en los puntos necesarios.

Sencillez

Este es, quizá, el objetivo del diseño más importante de todos y, sin duda, el más difícil de lograr en la práctica. Una estructura sencilla es fácil de entender, fácil de verificar, fácil de documentar y fácil de modificar. Para reducir una tarea compleja a un programa de estructura sencilla hacen falta conocimientos técnicos, práctica y paciencia, pero el resultado final siempre merece la pena el esfuerzo realizado.

11.3 Objetivos menos importantes

Cuando aparecieron los primeros lenguajes de alto nivel, los ordenadores eran menos potentes, más lentos y más caros proporcionalmente que los actuales y, en consecuencia, era de la mayor importancia que los programas fuesen cortos, porque así se reducía al mínimo el tiempo que empleaba el ordenador en traducirlos y ejecutarlos.

Sin embargo, la brevedad ya no resulta un objetivo útil. En el mundo actual de ordenadores rápidos, potentes y baratos, lo más costoso es el tiempo que emplean los programadores en desarrollar, corregir y mantener los programas. No es que los programas largos e ineficaces sean particularmente apreciados, pero en cualquier caso no hay duda de que los programas reducidos al mínimo no suelen ser correctos y además resultan casi imposibles de leer y mantener.

No obstante, hay casos en que sí conviene que un programa sea corto: cuando va a ejecutarse en un microordenador pequeño, por ejemplo. En tales situaciones, no hay que perder de vista los objetivos de corrección, claridad de estructura, facilidad de modificación y legibilidad durante el diseño.

11.4 Algunas limitaciones

Sería magnífico poder desarrollar los programas a voluntad hasta satisfacer completamente todos los objetivos de diseño, pero en la práctica eso no ocurre nunca. Casi todos los programas tienen una fecha límite de terminación y un presupuesto fijo, y es preciso acercarse lo más posible a los objetivos de diseño descritos sin salirse de esas limitaciones de tiempo y dinero.

11.5

Técnicas de diseño

Hemos hablado hasta ahora de lo que se persigue con el buen diseño: sencillez, corrección, claridad de estructura, facilidad de modificación y legibilidad, todo ello dentro de unas limitaciones de tiempo y dinero. Veremos ahora de qué forma pueden alcanzarse esos objetivos.

Actualmente se usan varias técnicas de diseño más o menos formales, de las que algunas exigen el uso de una notación especial. El método expuesto aquí es uno de los más frecuentes y menos formales, y no obliga a ninguna notación especial. Se llama **refinamiento por etapas.**

11.6

Refinamiento por etapas

Consiste básicamente en la descomposición de una tarea en varias etapas menores, hasta que cada una de éstas pueda programarse directamente. Partiendo de las especificaciones generales de la tarea que ha de ejecutarse, los sucesivos refinamientos expresan las fases de dicha tarea con un detalle creciente. Las etapas se describen en un estilo claro y breve, hasta que puedan expresarse en lenguaje de programación.

Pero el refinamiento por etapas tiene también sus trampas: en efecto, una tarea puede descomponerse de muchas formas, no todas ellas programables; si el proceso iniciado lleva a un callejón sin salida, hay que volver a empezar y repetir el proceso hasta dar con el camino acertado.

11.7

Diagrama de flujo

En lo que va de libro, hemos recurrido constantemente a los diagramas de flujo para ilustrar la corriente de control de programas y segmentos de programas, pero hay que advertir que no constituyen un instrumento muy útil a la hora de diseñar programas, porque obligan a introducir demasiados detalles en una fase muy temprana. Por tanto, en este capítulo, como en el resto del libro, los usaremos para ilustrar el "producto acabado". De hecho, los diagramas de flujo no se utilizan durante el diseño.

11.8

Ejemplos resueltos

Veremos ahora un par de ejemplos prácticos que ilustrarán la aplicación del refinamiento por etapas al diseño de programas. El primer ejemplo es un programa bastante corto escrito en un solo módulo. El otro refleja la aplicación de la técnica a un programa de varios módulos.

11.9

Programa ejemplo 11.1

Especificaciones del programa

Suponiendo que no disponga de la función raíz cuadrada, escriba un programa que la calcule con una precisión determinada. El grado de precisión es la diferencia entre el número y el cuadrado de su raíz cuadrada calculada por el programa.

Primer refinamiento

La descomposición inicial de la tarea es bastante obvia:

Introducir el número y el grado de precisión.
Verificar si el número es positivo.
Calcular la raíz cuadrada.
Mostrar el resultado.

Segundo refinamiento

Los pasos de entrada, verificación y salida son lo suficientemente evidentes como para programarlos. La fase de procesado exige un refinamiento mayor. La consulta a un libro de matemáticas proporciona la siguiente fórmula de cálculo de la raíz cuadrada:

Si s es el valor estimado de la raíz cuadrada de un número n, se obtiene una estimación más exacta s^+ mediante la fórmula

$$s^+ = \tfrac{1}{2}\left(s + \frac{n}{s}\right)$$

Si se usa esta fórmula, los pasos que hay que dar para calcular la raíz son los siguientes:

Elegir una estimación inicial de la raíz cuadrada.
Repetir el proceso:

Calcular una estimación más exacta mediante la fórmula citada.

Hasta alcanzar el grado de exactitud deseado.

Tercer refinamiento

Aunque puede ya escribirse un programa a partir de los pasos del segundo refinamiento, incluimos aquí una nueva fase en la que el mencionado proceso se expresa ya matemáticamente.

Elegir una estimación inicial de la raíz cuadrada:

La mitad del número constituye una estimación de exactitud suficiente

$$r = \frac{n}{r}$$

Repetir el proceso:

Calcular una estimación más exacta mediante la fórmula

$$r^+ = \tfrac{1}{2}\left(r + \frac{n}{r}\right)$$

(En el programa puede usarse el mismo nombre de la variable en ambos miembros de la ecuación.)

Hasta alcanzar el grado de exactitud deseado,

es decir, hasta que el valor absoluto de $(n - s^2)$ sea menor que a, siendo a el grado de exactitud.

Variables

N Número.
P Grado de precisión.
R Raíz cuadrada.

Diagrama de flujo

La figura 11.1 recoge el flujo de control del programa.

Programa

El programa se ha escrito a partir de los pasos en que acaba de descomponerse la tarea. Hay que insistir en que el proceso de diseño seguido hasta este punto es independiente del lenguaje de programación en que se trabaje.

```
100 REM PROGRAMA EJEMPLO 11.1
105 REM PROGRAMA QUE CALCULA LA RAIZ CUADRADA
110 REM DE UN NUMERO, DADO EL
115 REM GRADO DE PRECISION
120 REM
125 REM ENTRADA DEL NUMERO N Y DEL GRADO DE
130 REM PRECISION P
135 INPUT N, P
140 REM VERIFICA QUE EL NUMERO ES POSITIVO
145 IF N >= 0 THEN 160
150 PRINT "NUMERO NEGATIVO, NO TIENE RAIZ CUADRA
           DA"
155 STOP
160 REM CALCULO INICIAL ESTIMATIVO DE LA RAIZ
        CUADRADA
165 LET R = N/2
170 REM REPETIR
175 REM ESTIMACION DE CALCULO MAS APROXIMADO
180 LET R = 1/2 * (R + N/R)
185 REM HASTA QUE LA ACTUAL ESTIMACION
190 REM SEA LO BASTANTE PRECISA
195 IF ABS(N - R*R) > ABS(A) THEN 180
200 REM SALIDA DE RESULTADOS
205 PRINT "RAIZ CUADRADA:"; R
210 END
```

Puntos de interés

- Se ha "dado la vuelta" a la condición de repetición para simplificar la lógica del programa.
- Las notas, quizá demasiado numerosas, hacen muy legible el programa.

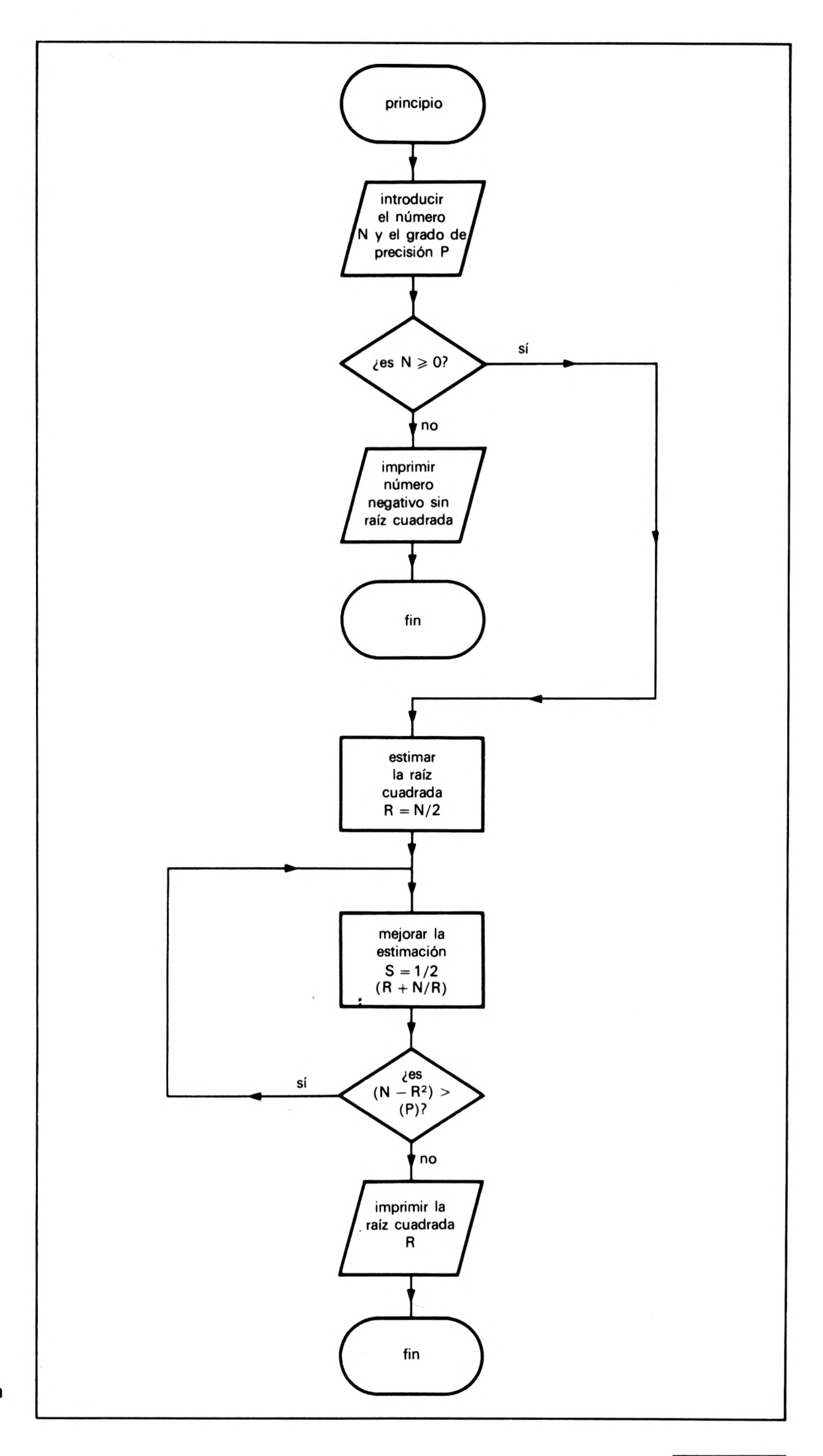


Figura 11.1
Diagrama de flujo del programa ejemplo 11.1

11.10

Programa ejemplo 11.2

Especificaciones del programa

El objetivo del programa es mantener una lista actualizada de los códigos de identificación de todos los aviones de un aeropuerto. El programa debe responder a las órdenes siguientes:

INTRODUCIR ID	Se añade a la lista el avión con la identificación **ID**.
SACAR ID	El avión de identificación **ID** se borra de la lista; si la identificación no se encuentra, se produce un mensaje.
INDAGAR ID	Se busca en la lista para tratar de encontrar el avión de identificación **ID**. Se produce un mensaje que informa del resultado de la búsqueda.
VISUALIZAR	Se produce una relación completa de las identificaciones de todos los aparatos.
DETENER	El programa se detiene.

Primer refinamiento

Dado que, evidentemente, el programa ha de estructurarse en varios módulos, la primera etapa del proceso de refinamiento será decidir esa estructura general, es decir, la forma en que se organiza el programa a partir de sus módulos constituyentes.

Tras alguna reflexión, se llega a una estructura de tres capas o **niveles,** muy usada en otras partes de este libro. En el nivel superior se encuentra el **módulo de control,** encargado de identificar las diversas órdenes. Según la orden recibida, este módulo transfiere el control a uno de los **módulos de operación.** Estos constituyen el nivel central de la estructura. El nivel inferior está formado por una serie de **módulos de servicio** que interaccionan directamente con la estructura de datos utilizada para almacenar la información.

La figura 11.2 ilustra la estructura descrita. Conviene insistir en que se trata de una estructura muy frecuente.

Segundo refinamiento

Decidida la estructura general del programa, hay que especificar las estructuras de datos que van a usarse y esbozar el funcionamiento de cada uno de los módulos.

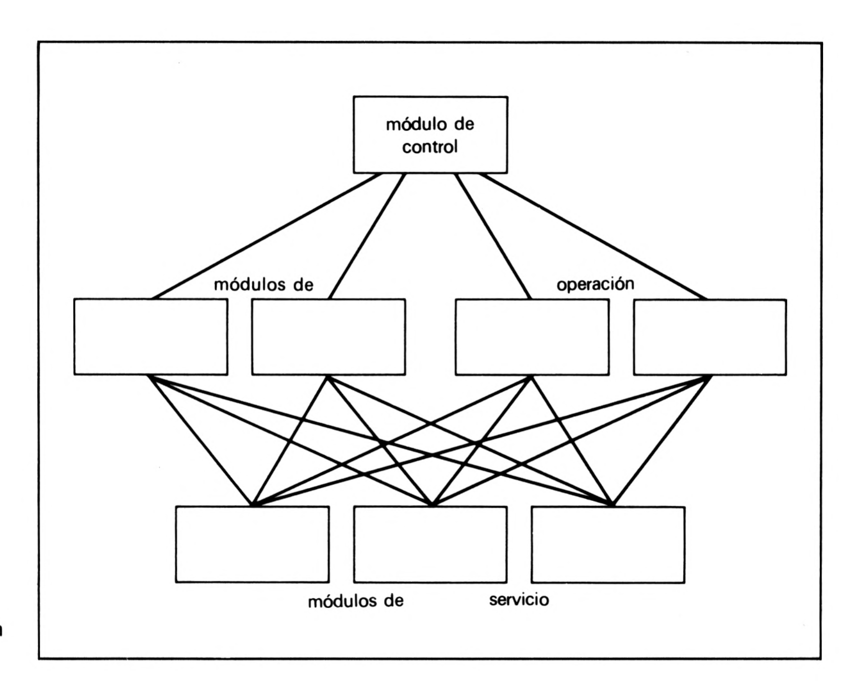


Figura 11.2
Estructura general del programa ejemplo 11.2

Una **matriz** se adapta bien a los fines de este programa, y en el capítulo 20 se describe otra estructura de datos que también podría servir. Como el tamaño de una matriz queda fijado en el mismo momento en que se declara y el programa obliga a almacenar una cantidad de datos variable, hay que utilizar un **indicador de elemento vacío;** para ello sirve la cadena alfanumérica "**XXXXXX**", que tiene la misma longitud que el código de identificación de un avión, pero a la vez no puede confundirse con ninguna identificación real.

Una matriz puede considerarse como un mapa dispuesto entre un índice y el elemento localizado por ese índice. Esto sugiere los siguientes módulos de servicio:

CARGAR un elemento dado en la matriz en una posición de índice dada.
ACCEDER al elemento de la matriz en una posición de índice dada.
BUSCAR la matriz a partir del índice de un elemento dado. Si no se encuentra el elemento, volver el índice al valor 0.

Con estos módulos de servicio, los módulos de operación serían:

INICIALIZAR la matriz, cargando todas las posiciones con marcadores de elementos vacíos.
INTRODUCIR ID. Buscar la matriz hasta localizar un indica-

dor de elemento vacío. Si se encuentra alguno, cargar esa posición con la identificación de un aparato y presentar un mensaje de confirmación. Si no hay elementos vacíos, anunciarlo mediante un mensaje apropiado.

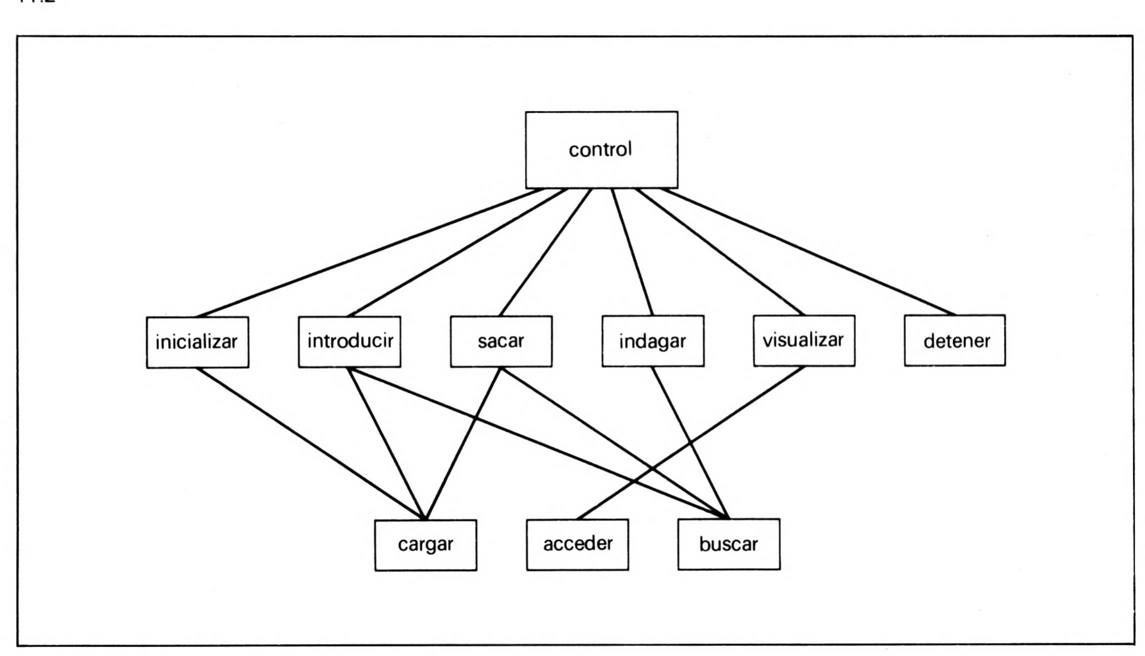
SACAR ID. Buscar la matriz hasta localizar la identificación deseada. Si se encuentra, cargar en su posición un indicador de elemento vacío y producir un mensaje de confirmación. Si la identificación no se localiza, anunciarlo mediante un mensaje apropiado.

INDAGAR ID. Buscar la matriz hasta localizar la identificación deseada. Si se encuentra, confirmarlo mediante un mensaje; en caso contrario, el mensaje advertirá que el aparato en cuestión no está presente.

VISUALIZAR. Acceder uno por uno a todos los elementos de la matriz y presentar a la salida todos los que no contengan indicadores de posición vacía.

DETENER. Interrumpir el programa y presentar un mensaje adecuado.

Figura 11.3
Módulos del programa ejemplo 11.2



La figura 11.3 ilustra las interrelaciones entre los diversos módulos. El de control es el programa principal y los demás son subprogramas.

Nota sobre los datos

La identificación de un avión consta de dos letras seguidas por un guión y un segundo grupo de tres letras. Por ejemplo:

PH-BUL.
DM-FIY.

Variables

Global:

A$(30) Matriz de identificaciones de los aparatos. Utilizada por los tres módulos de servicio.

En los módulos de servicio:

CARGAR:	Parámetros	**L$**	Elemento para cargar.
		I	Posición en que se ha de cargar el elemento.
ACCEDER:	Parámetros	**J**	Posición del elemento deseado.
		M$	Elemento al que se accede.
BUSCAR:	Parámetros	**N$**	Elemento para cargar.
		K	Posición del elemento.
	Local	**L**	Contador de bucle.

En los módulos de operación:

INICIALIZAR:	Local	**N**	Contador de bucle.
INTRODUCIR:	Parámetros	**D$**	Identificación del aparato.
		C	Variable de verificación.
SACAR:	Parámetros	**D$**	Identificación del aparato.
		C	Variable de verificación.
INDAGAR:	Parámetros	**D$**	Identificación del aparato.
		C	Variable de verificación.
VISUALIZAR:	Local	**P**	Contador de bucle.
	Parámetros	**C**	Variable de verificación.

En el módulo de control:

Locales	**O$**	Orden.
	E	Número de caracteres de la orden.

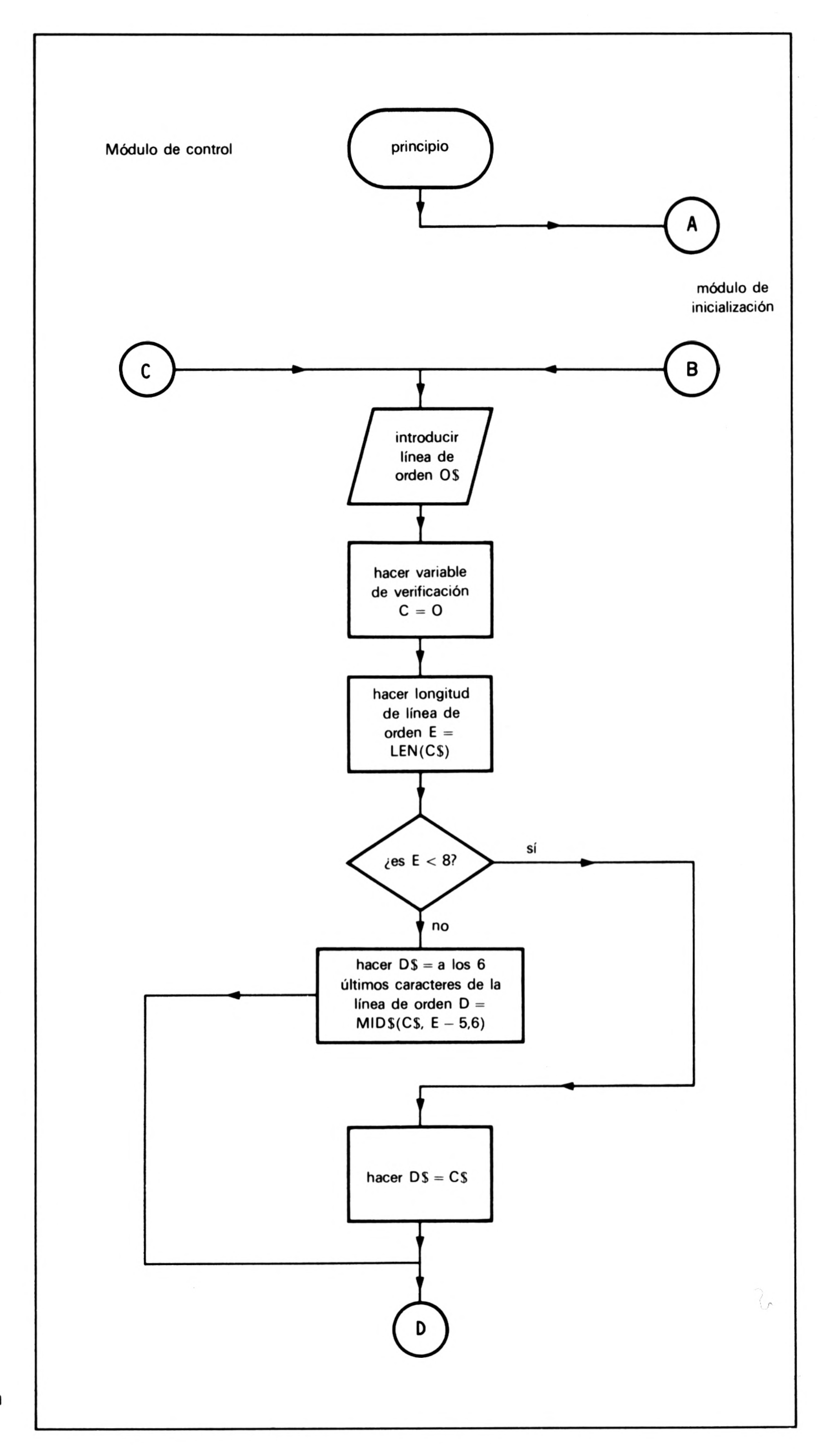
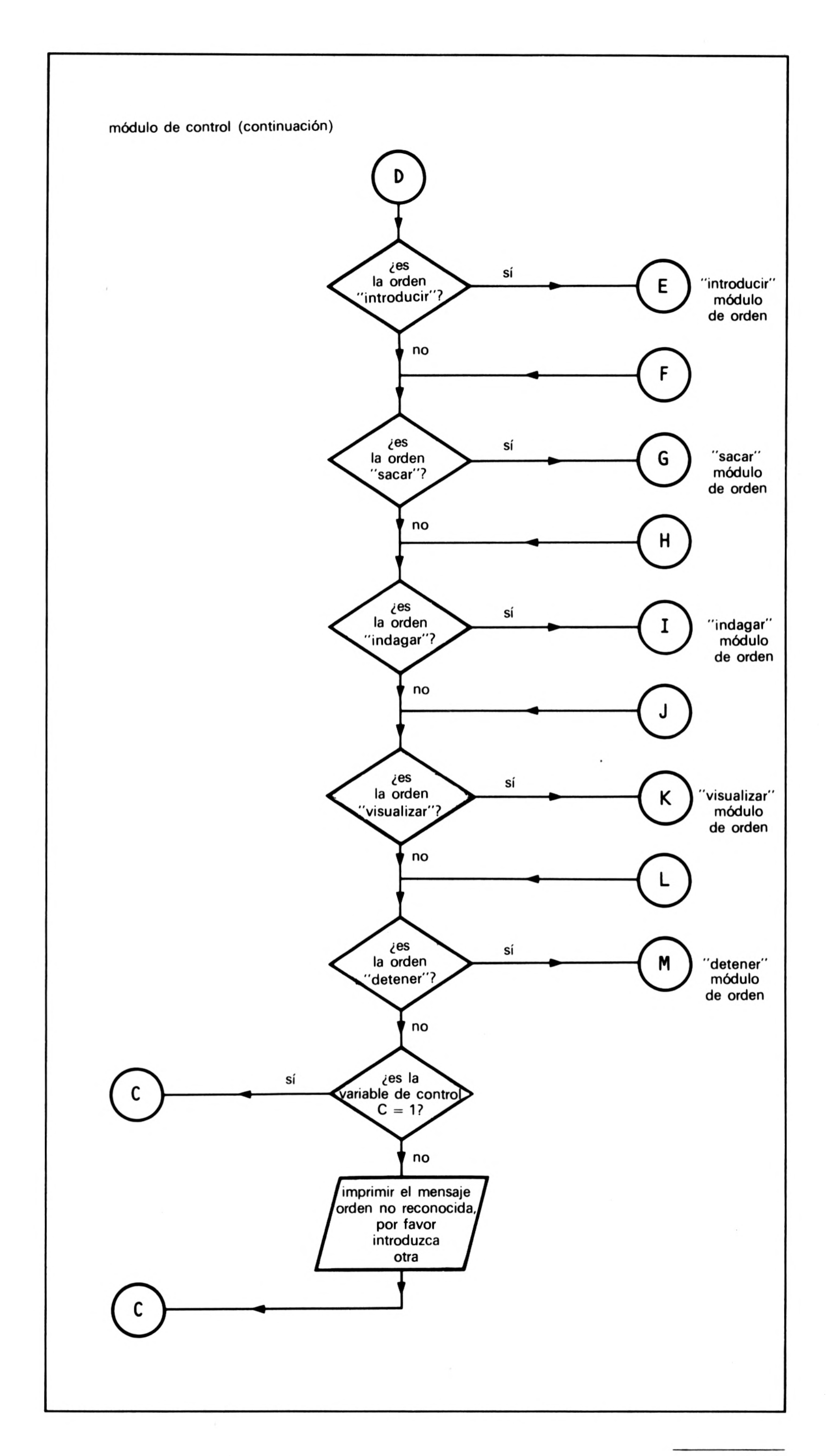
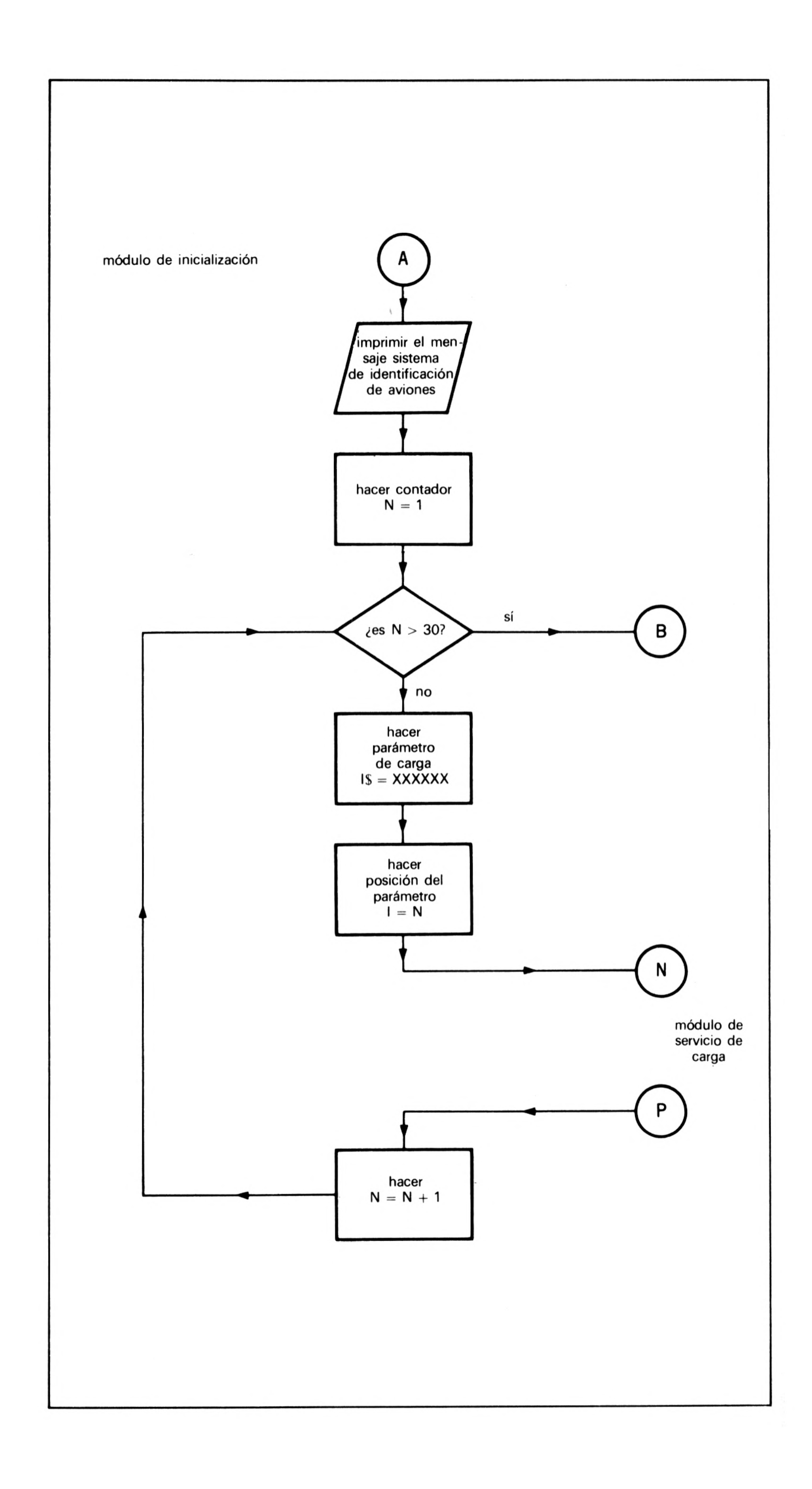
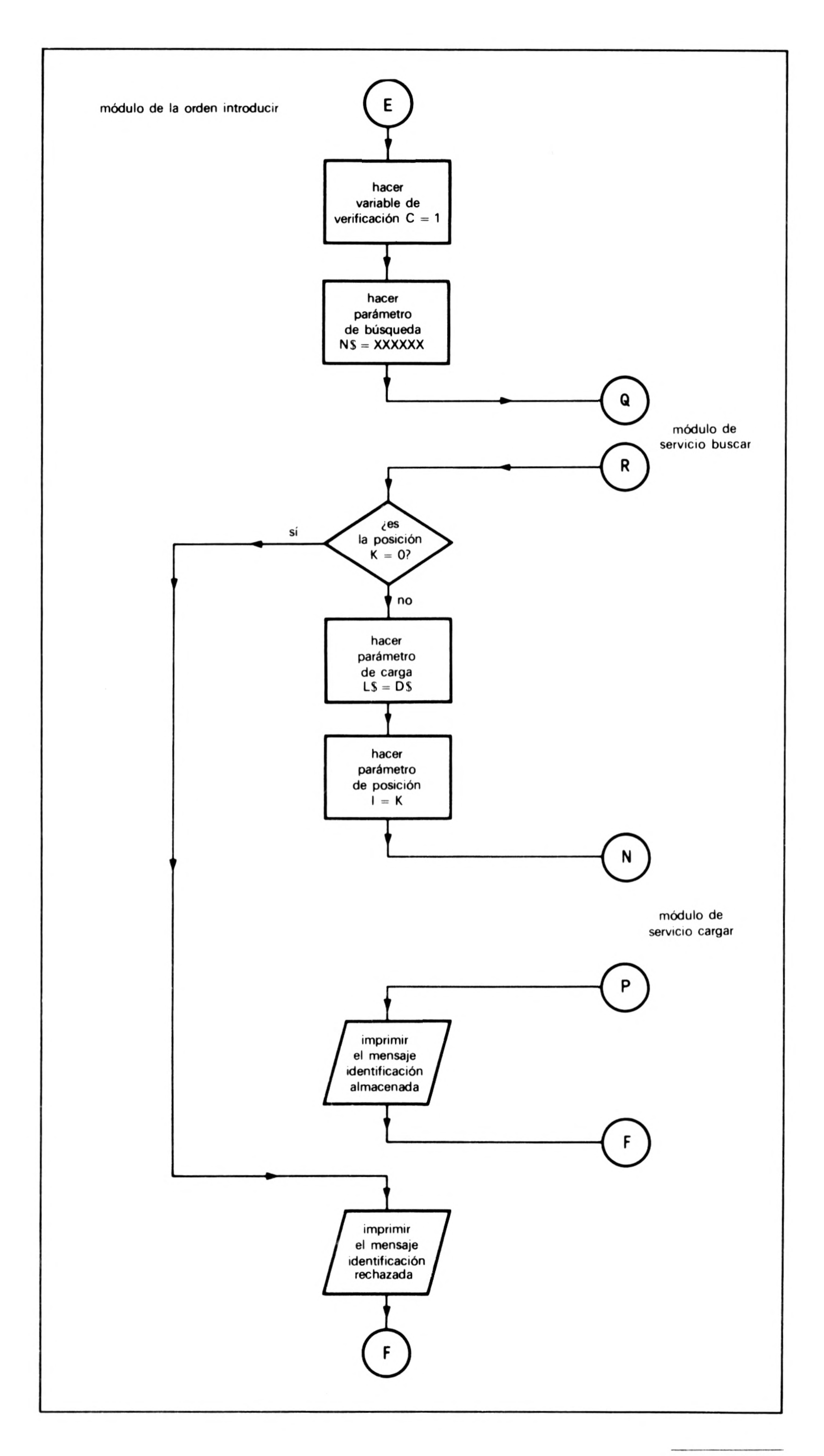
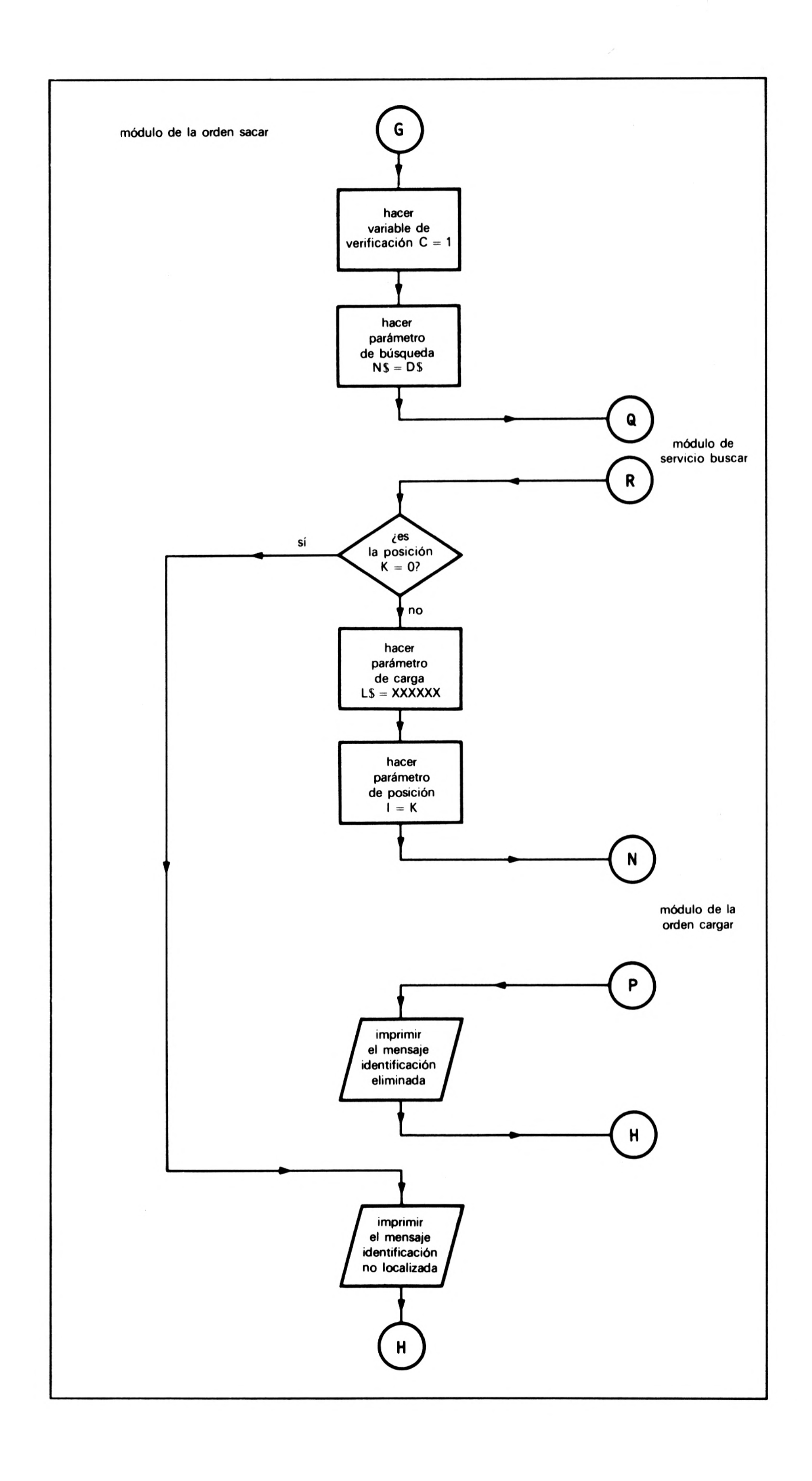


Figura 11.4
Diagrama de flujo del programa ejemplo 11.2

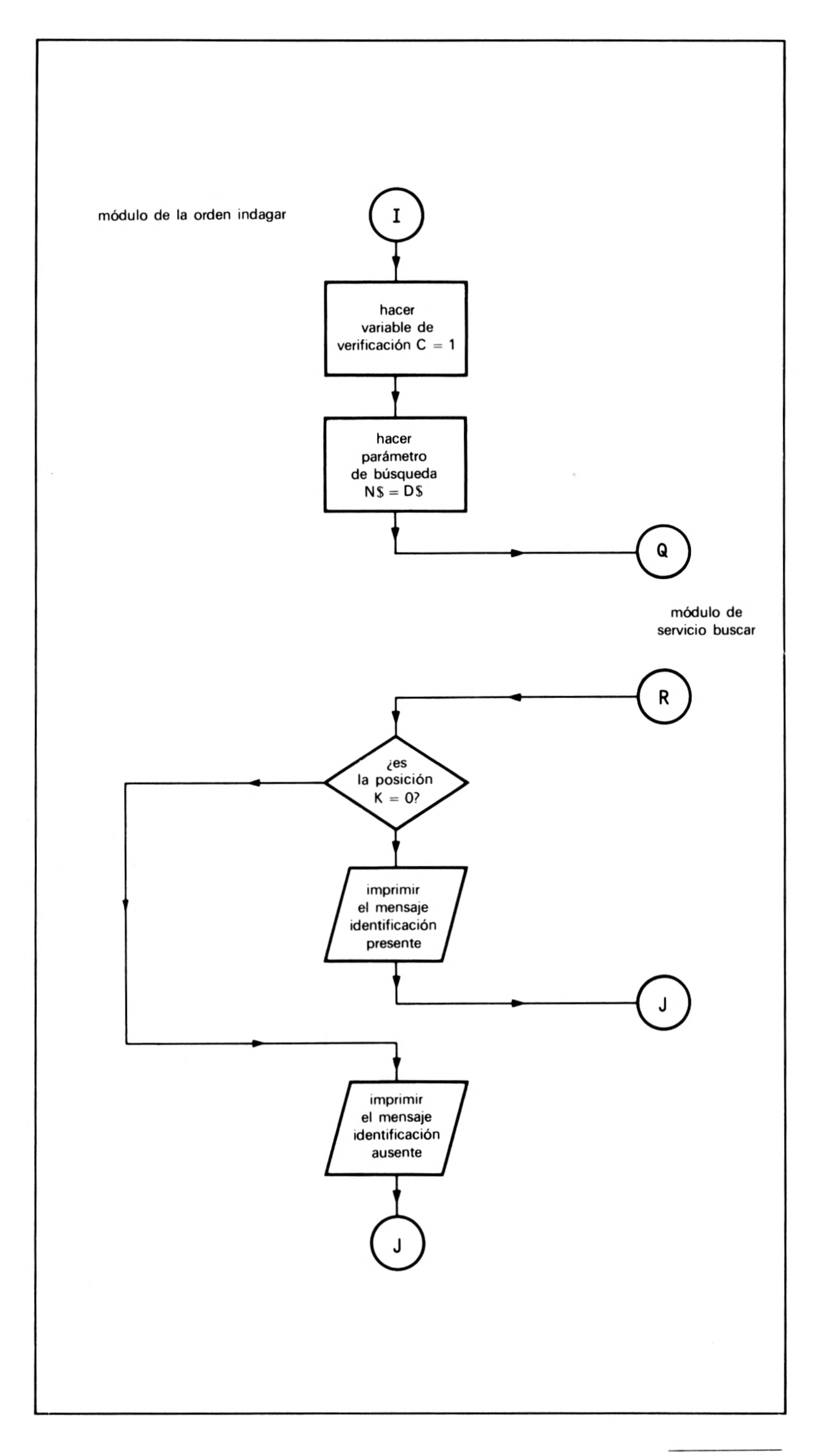

módulo de control (continuación)
D
¿es
la orden
"introducir"?
sí
E
"introducir"
módulo
de orden
no
F
¿es
la orden
"sacar"?
sí
G
"sacar"
módulo
de orden
no
H
¿es
la orden
"indagar"?
sí
I
"indagar"
módulo
de orden
no
J
¿es
la orden
"visualizar"?
sí
K
"visualizar"
módulo
de orden
no
L
¿es
la orden
"detener"?
sí
M
"detener"
módulo
de orden
no
¿es la
variable de control
C = 1?
sí
C
no
imprimir el mensaje
orden no reconocida,
por favor
introduzca
otra
C

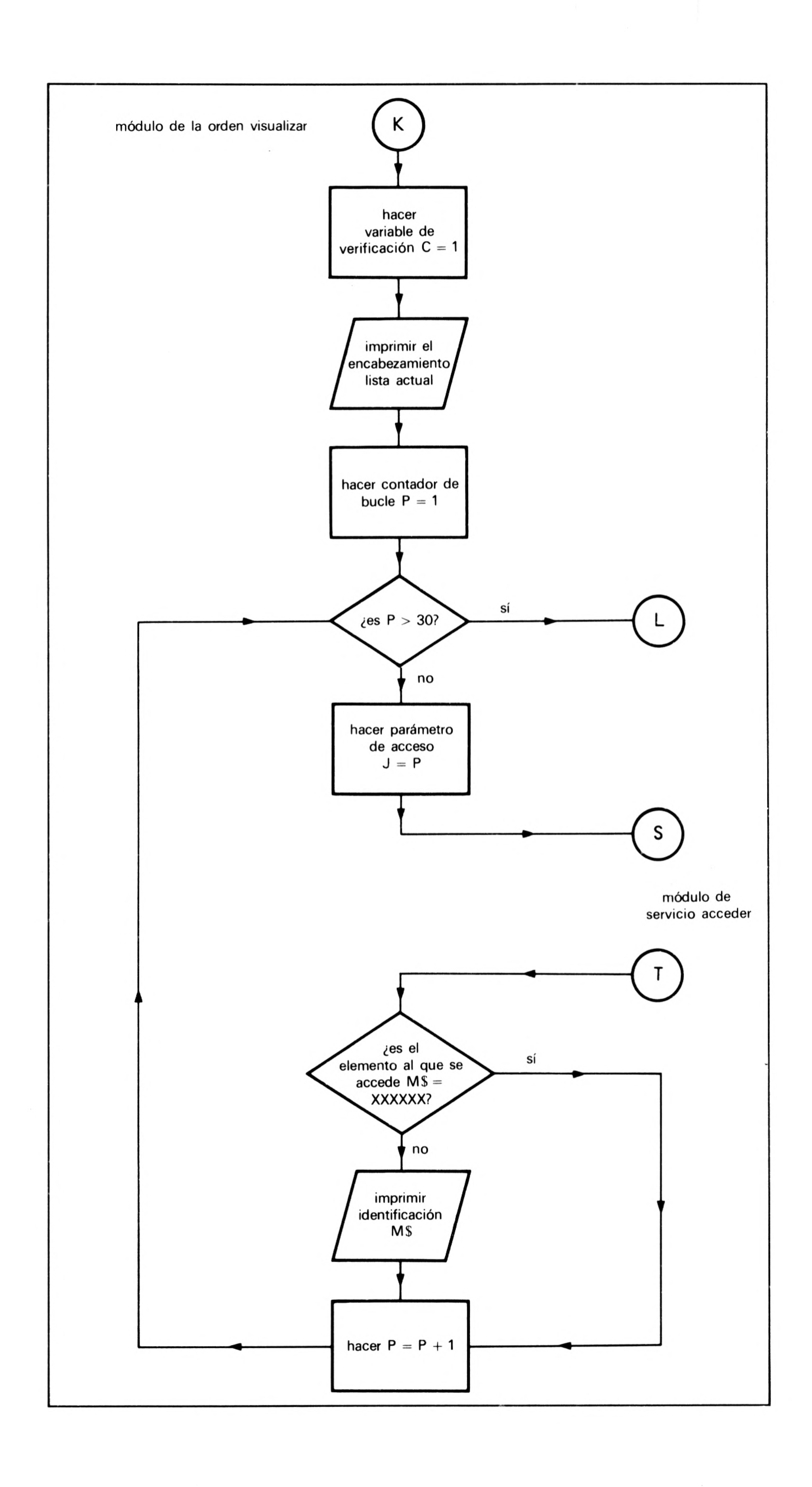

módulo de inicialización
A
imprimir el mensaje sistema de identificación de aviones
hacer contador N = 1
¿es N > 30?
sí
B
no
hacer parámetro de carga I$ = XXXXXX
hacer posición del parámetro I = N
N
módulo de servicio de carga
P
hacer N = N + 1


módulo de la orden introducir
E
hacer
variable de
verificación C = 1
hacer
parámetro
de búsqueda
N$ = XXXXXX
Q
módulo de
servicio buscar
R
¿es
la posición
K = 0?
sí
no
hacer
parámetro
de carga
L$ = D$
hacer
parámetro
de posición
I = K
N
módulo de
servicio cargar
P
imprimir
el mensaje
identificación
almacenada
F
imprimir
el mensaje
identificación
rechazada
F



módulo de la orden sacar
G
hacer
variable de
verificación C = 1
hacer
parámetro
de búsqueda
N$ = D$
Q
módulo de
servicio buscar
R
¿es
la posición
K = 0?
sí
no
hacer
parámetro
de carga
L$ = XXXXXX
hacer
parámetro
de posición
I = K
N
módulo de la
orden cargar
P
imprimir
el mensaje
identificación
eliminada
H
imprimir
el mensaje
identificación
no localizada
H


módulo de la orden indagar
I
hacer
variable de
verificación C = 1
hacer
parámetro
de búsqueda
N$ = D$
Q
módulo de
servicio buscar
R
¿es
la posición
K = 0?
imprimir
el mensaje
identificación
presente
J
imprimir
el mensaje
identificación
ausente
J



módulo de la orden visualizar
K
hacer
variable de
verificación C = 1
imprimir el
encabezamiento
lista actual
hacer contador de
bucle P = 1
¿es P > 30?
sí
L
no
hacer parámetro
de acceso
J = P
S
módulo de
servicio acceder
T
¿es el
elemento al que se
accede M$ =
XXXXXX?
sí
no
imprimir
identificación
M$
hacer P = P + 1

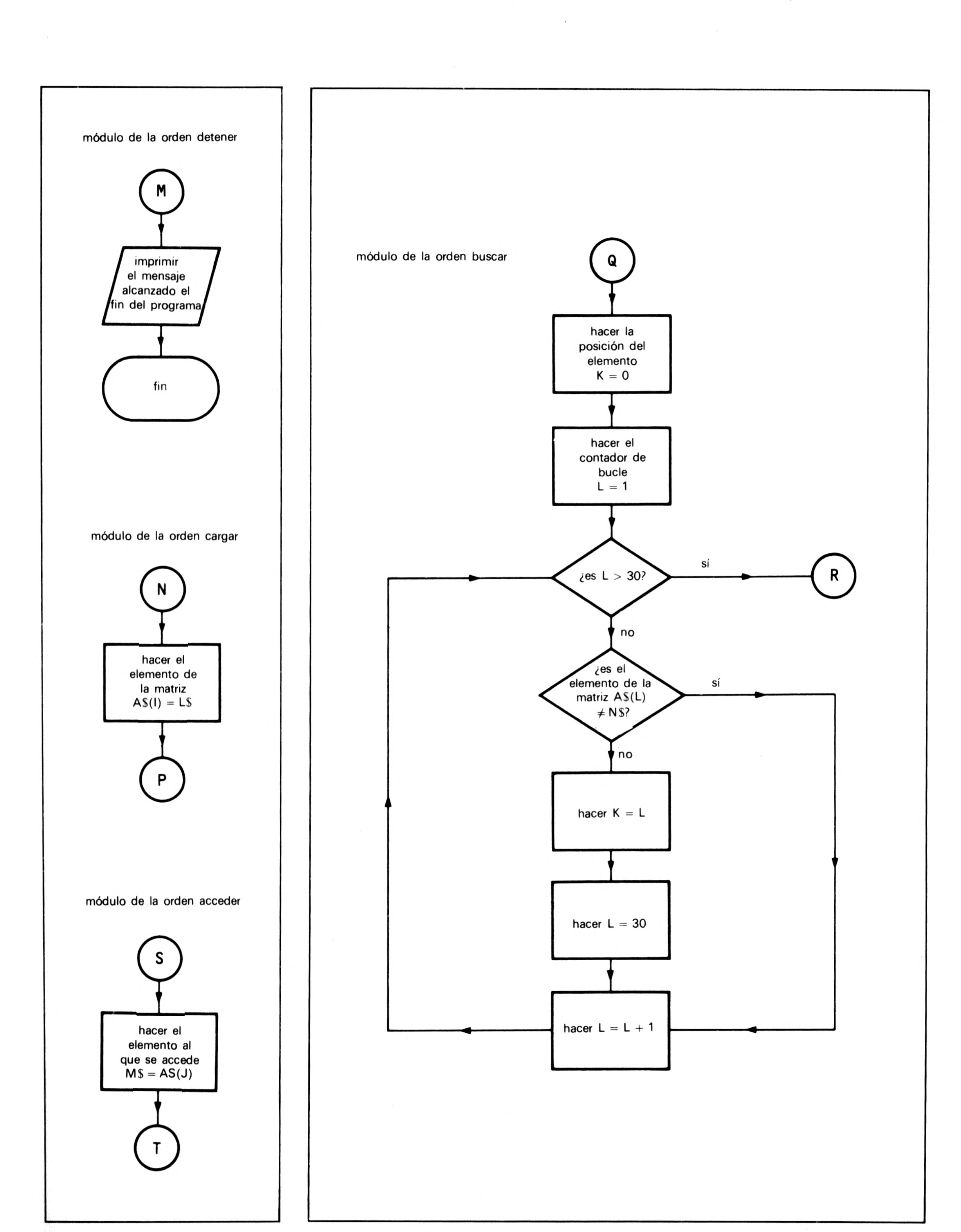

módulo de la orden detener
M
imprimir
el mensaje
alcanzado el
fin del programa
fin
módulo de la orden cargar
N
hacer el
elemento de
la matriz
A$(I) = L$
P
módulo de la orden acceder
S
hacer el
elemento al
que se accede
M$ = AS(J)
T
módulo de la orden buscar
Q
hacer la
posición del
elemento
K = 0
hacer el
contador de
bucle
L = 1
¿es L > 30?
sí
R
no
¿es el
elemento de la
matriz A$(L)
≠ N$?
sí
no
hacer K = L
hacer L = 30
hacer L = L + 1

Diagrama de flujo

La figura 11.4 ilustra el flujo de control en los diferentes módulos del programa ejemplo 11.2.

Programa

Los módulos del programa están ya especificados con suficiente detalle como para ser codificados en Basic. Se han introducido algunos comentarios adicionales mediante las oportunas instrucciones **REM**.

Módulo de control

```
1000 REM PROGRAMA EJEMPLO 11.2
1005 REM MANTENIMIENTO ACTUALIZADO DE UNA LISTA
1010 REM DE LOS CODIGOS DE UN AVION
1015 DIM A$(30): REM VARIABLE GLOBAL, IDENTIFICA
                     CIONES DEL AVION
1020 REM
1100 GOSUB 2000: REM INICIALIZACION DE LA MATRIZ
1105 REM
1200 REM BUCLE DE CONTROL PRINCIPAL
1205 INPUT "ORDEN?", O$
1210 LET C = 0
1215 LET E = LEN(C$)
1220 IF E < 8 THEN 1235
1225 LET D$ = MID$ (O$, E - 5, 6): REM AVION
1230 GOTO 1240
1235 LET D$ = O$
1240 IF MID$ (O$, 1, 10)= "INTRODUCIR" THEN GOSUB
     3000
1245 IF MID$ (O$, 1, 5)= "SACAR" THEN GOSUB 4000
1250 IF MID$ (O$, 1, 7)= "INDAGAR" THEN GOSUB
     5000
1255 IF MID$ (O$, 1, 10)= "VISUALIZAR" THEN
     GOSUB 6000
1260 IF MID$ (O$, 1, 7)= "DETENER" THEN
     GOSUB 7000
1265 IF C = 1 THEN 1200
1270 PRINT "ORDEN NO RECONOCIDA"
1275 PRINT "POR FAVOR INTRODUZCA OTRA"
1280 GOTO 1200
1285 REM
```

Módulo de inicialización

```
2000 REM MODULO DE INICIALIZACION
2005 PRINT "SISTEMA DE IDENTIFICACION DEL AVION"
2010 PRINT
2015 FOR N = 1 TO 30
```

```
2020 LET L$ = "XXXXXX"
2025 LET I = N
2030 GOSUB 8000: REM BORRAR ELEMENTO EN LA
                     POSICION N
2035 NEXT N
2040 RETURN
2045 REM
```

Módulo de la orden INTRODUCIR

```
3000 REM MODULO DE LA ORDEN "INTRODUCIR"
3005 REM PARAMETROS
3010 REM D$ : IDENTIFICACION DEL AVION
3015 REM C: VARIABLE DE COMPROBACION
3020 LET C = 1
3025 LET N$ = "XXXXXX"
3030 GOSUB 9000: REM BUSCA ELEMENTO EN BLANCO
3035 IF K = 0 THEN 3070
3040 REM BUSQUEDA CON EXITO
3045 LET L$ = D$
3050 LET I = K
3055 GOSUB 8000: REM CARGAR D$ EN LA POSICION K
3060 PRINT "IDENTIFICACION "; D$; " ALMACENADA"
3065 RETURN
3070 REM BUSQUEDA INFRUCTUOSA
3075 PRINT "ALMACENAMIENTO LLENO, IDENTIFICACION ";
           D$; " RECHAZADA"
3080 RETURN
3085 REM
```

Módulo de la orden SACAR

```
4000 REM MODULO DE LA ORDEN "SACAR"
4005 REM PARAMETROS
4010 REM D$: IDENTIFICACION DEL AVION
4015 REM C: VARIABLE DE COMPROBACION
4020 LET C = 1
4025 LET N$ = D$
4030 GOSUB 9000: REM BUSQUEDA DE D$
4035 IF K = 0 THEN 4070
4040 REM BUSQUEDA CON EXITO
4045 LET L$ = "XXXXXX"
4050 LET I = K
4055 GOSUB 8000: REM BORRAR ELEMENTO EN LA
                     POSICION K
4060 PRINT "IDENTIFICACION "; D$; " AUSENTE"
4065 RETURN
4070 REM BUSQUEDA INFRUCTUOSA
4075 PRINT "IDENTIFICACION "; D$; " NO ENCONTRADA"
4080 RETURN
4085 END
```

Módulo de la orden INDAGAR

```
5000 REM MODULO DE LA ORDEN "INDAGAR"
5005 REM PARAMETROS
5010 REM D$: IDENTIFICACION DEL AVION
5015 REM C: VARIABLE DE COMPROBACION
5020 LET C = 1
5025 LET N$ = D$
5030 GOSUB 9000: REM BUSQUEDA DE D$
5035 IF K = 0 THEN 5055
5040 REM BUSQUEDA AFORTUNADA
5045 PRINT "IDENTIFICACION "; D$; " PRESENTE"
5050 RETURN
5055 REM BUSQUEDA INFRUCTUOSA
5060 PRINT "IDENTIFICACION "; D$; " NO ESTA PRE
          SENTE"
5065 RETURN
5070 REM
```

Módulo de la orden VISUALIZAR

```
6000 REM MODULO DE LA ORDEN "VISUALIZAR"
6005 REM PARAMETROS
6010 REM C: VARIABLE DE COMPROBACION
6015 LET C = 1
6020 PRINT "LISTA GENERAL"
6025 FOR P = 1 TO 30
6030 LET J = P
6035 GOSUB 10000 : REM ACCEDER AL ELEMENTO EN LA
                       POSICION J
6040 IF M$ = "XXXXXX" THEN 6050
6045 PRINT M$
6050 NEXT P
6055 RETURN
6060 REM
```

Módulo de la orden DETENER

```
7000 REM MODULO DE LA ORDEN "DETENER"
7005 PRINT "SE HA ALCANZADO EL FINAL DEL PROGRAMA
          "
7010 STOP
7015 REM
```

Módulo de servicio CARGAR

```
8000 REM INTRODUCCION DEL ELEMENTO EN UNA MATRIZ
8005 REM PARAMETROS
8010 REM L$: ELEMENTO QUE VA A SER INTRODUCIDO
8015 REM I: POSICION EN LA QUE EL ELEMENTO VA A
          SER INTRODUCIDO
```

```
8020 LET A$(I) = L$
8025 RETURN
8030 REM
```

Módulo de servicio ACCEDER

```
9000 REM ACCESO DEL ELEMENTO A LA MATRIZ
9005 REM PARAMETROS
9010 REM M$: ELEMENTO
9015 REM J: POSICION DEL ELEMENTO REQUERIDO
9020 LET M$ = A$(J)
9025 RETURN
9030 REM
```

Módulo de servicio BUSCAR

```
10000 REM BUSQUEDA DEL ELEMENTO EN LA MATRIZ
10005 REM PARAMETROS
10010 REM N$: ELEMENTO QUE SE VA A LOCALIZAR
10015 REM K: POSICION DEL ELEMENTO
10020 LET K = 0: REM SE SUPONE NO HALLADO
                     INICIALMENTE
10025 FOR L = 1 TO 30
10030 IF A$(L) = N$ THEN 1045
10035 LET K = L
10040 LET L = 30
10045 NEXT L
10050 RETURN
10055 REM
```

Puntos de interés

- La estructuración del programa en módulos se ha realizado ateniéndose rigurosamente a los siguientes principios:
  - — La primera línea de cada módulo empieza por un número múltiplo exacto de 1.000.
  - — La entrada a los diversos módulos sólo puede producirse por la línea de principio.
  - — Las bifurcaciones internas de un módulo llevan a cualquier línea de ese módulo.
  - — Las bifurcaciones de salida de un módulo llevan únicamente a la línea de principio de otro.
  - — Aparte el de control, todos los módulos están escritos como subprogramas.

  Con ligeras modificaciones, estos principios de estructuración de programas se emplean igual en el resto del libro.
- La variable de verificación **C** determina si la orden introducida es correcta. Se pone a cero (indicación de orden incorrecta) en el

programa principal, y es un parámetro para todos los subprogramas de órdenes. Si el control pasa a cualquiera de ellos, **C** se convierte en 1 dentro del mismo para indicar orden correcta. **C** se verifica al final del bucle de control principal.

- La variable **D$** representa los últimos seis caracteres de una orden, es decir, la identificación de un aparato. Si la orden tiene menos de ocho caracteres, **D$** se fija a la serie completa de los mismos.
- Aunque el módulo **DETENER** es un subprograma, el control no vuelve desde él, porque contiene una instrucción **STOP**.
- El control sale del bucle del módulo de servicio **BUSCAR** por el método expuesto en la sección 5.7.

11.11 Conclusión

Los dos ejemplos prácticos de este capítulo han servido para ilustrar la técnica de diseño de programas conocida como refinamiento por etapas. Hay que insistir en que se trata de un método sencillo e informal, apropiado para tareas de programación muy diversas. El número de etapas de refinamiento y el detalle con que se expresen las mismas antes de escribir el programa dependen de la tarea concreta y de la experiencia del programador. El método puede resumirse como sigue:

- Partir de una exposición clara de la tarea que ha de programarse.
- Decidir la estructura general del programa, es decir, su descomposición en módulos.
- Especificar la función de cada módulo en términos generales, junto con la relación que guardan con el resto del programa.
- Decidir las variables y las estructuras de datos que van a usarse y exponer las fases de cada módulo con mayor detalle.
- Repetir el proceso anterior hasta que el detalle baste para redactar el programa.
- Si en cualquier etapa se llega a un callejón sin salida, retroceder una o más etapas y probar con una descomposición diferente de la tarea.
- Mantener una estructura general lo más sencilla posible.

Este método de diseño es el usado para la mayoría de los programas propuestos en el resto del libro, y es muy recomendable aplicarlo, directamente o con alguna modificación, a la creación de sus propios programas. Es importante comprobar en todas las fases del

diseño y la escritura de un programa que se cumplen los objetivos mencionados al principio del capítulo, y la técnica de diseño propuesta contribuye eficazmente a ello.

11

Ejercicio

1. Resuma los objetivos que debe cumplir un programa bien diseñado.
2. Redacte una lista con algunas de las ventajas de los programas estructurados en módulos.
3. El refinamiento por etapas es un método de diseño que va de "arriba a abajo". Explique lo que entiende en este contexto por ir de "arriba a abajo". Esboce en líneas generales lo que haría para diseñar un programa avanzando de "abajo a arriba".
4. Un programa tiene las siguientes especificaciones:

> Introducir los elementos de dos matrices de 3 por 3.
> Calcular la suma de dichas matrices, y
> Mostrar el resultado.

Una primera descomposición de esa tarea en pasos menores sería:

> Introducir los elementos de la primera matriz.
> Introducir los elementos de la segunda matriz.
> Calcular la matriz suma.
> Mostrar la matriz suma.

a) Exponer estas etapas con más detalle y escribir un programa que las ejecute.
b) Descomponer la misma tarea en etapas diferentes, expresarlas con más detalle y escribir por fin un programa.
c) Comparar los dos programas con respecto a los objetivos de diseño.

5. Diseñar, siguiendo el método del refinamiento por etapas, un programa que satisfaga las siguientes especificaciones:

> Introducir una cadena de caracteres alfanuméricos. Presentar a la salida todos los caracteres junto con el número de veces que cada uno aparece repetido en la cadena alfanumérica.

6. Un programa tiene las siguientes especificaciones:

> Se trata de procesar los resultados obtenidos por un grupo de alumnos; para ello:
>
> Se introduce el nombre de cada uno de los alumnos seguido de una serie de porcentajes. El conjunto termina con el valor −1. Se calcula y se muestra la media de los porcentajes.
>
> Si el porcentaje no es correcto, es decir, si no es un entero comprendido entre 0 y 100, se solicita la introducción de otro mediante el correspondiente mensaje.
>
> La entrada termina con la cadena de caracteres alfanuméricos **XXX**, y acto seguido aparece en la salida la media ponderada de todos los

alumnos. La media ponderada es la suma total de todos los porcentajes de los alumnos dividido por el número total de porcentajes del grupo completo. Por tanto, no se obtiene sumando las medias y dividiendo el resultado por el número de alumnos.

La tarea expuesta podría descomponerse en los siguientes módulos:

Módulo de entrada y suma: se encarga de introducir el nombre de un alumno y su conjunto de porcentajes. Suma los porcentajes correctos al total del alumno y acumula el número de porcentajes.

Módulo de media y salida: calcula y lleva a la salida la media de cada alumno y actualiza la suma total y el número de porcentajes.

Módulo de convalidación: verifica si un número dado es un entero comprendido entre 0 y 100.

a) Expresar más detalladamente las etapas de cada módulo. Especificar también la relación que hay entre los módulos.
b) Elegir un conjunto adecuado de variables y escribir un programa a partir de las etapas detalladas. No es necesario crear matrices ni estructuras de datos similares.
c) Comentar la descomposición del programa en módulos.
d) Evaluar el programa en términos de los objetivos de diseño.

7. Modificar el programa ejemplo 11.2 para incluir un **módulo de validación.** Si la orden contiene la identificación de un aparato (es decir, si tiene más de 8 caracteres), se pasan los seis últimos al módulo de validación, que verifica si el tercer carácter es un guión y todos los demás letras mayúsculas. Siga la técnica de refinamiento por etapas para diseñar el nuevo módulo.

8. Diseñar y escribir un conjunto de programas para crear, actualizar y acceder a uno o más ficheros contables. Cada uno de los registros de los ficheros contiene la información relativa a una transacción: fecha, tipo de transacción, detalles de la misma, importe y saldo de la cuenta. Las cuentas pueden ser personales, de negocios o de una escuela o una asociación juvenil.


RESET
RETURN

12 Comprobación de programas

Este capítulo se refiere al proceso decisivo de comprobar o verificar un programa una vez escrito. Primero se estudian los objetivos de dicho proceso de comprobación, tras lo cual se exponen algunas técnicas para llevarlo a cabo. El empleo de estas técnicas se realizará sobre los programas que han servido de ejemplo en capítulos anteriores.

El material de este capítulo está estrechamente relacionado con el del último, dedicado al diseño de programas, y muchas de las observaciones generales referentes a un programa bien diseñado serán de aplicación aquí.

También guarda relación con los capítulos de documentación y modificación de programas y, en cierto modo, con todos los del resto del libro, ya que cabe suponer que las técnicas aprendidas aquí se pondrán en práctica al estudiar otros temas del texto y cada vez que se realice algún trabajo de programación. Las ideas que van a tratarse a continuación son de tipo general, independientes de cualquier lenguaje de programación concreto.

12.1

Objetivos de la comprobación de programas

Hablando en términos generales, un programa se comprueba en primer lugar para evitar errores o para detectar la causa de los ya cometidos, y en segundo lugar para ver en qué medida el programa terminado satisface las especificaciones que motivaron su creación.

Los errores de programación son de dos tipos: **sintácticos** y de **ejecución.** Los de sintaxis se deben al empleo incorrecto del lenguaje de programación y se detectan en la fase de compilación del programa. Por lo general se localizan mediante mensajes de error que comunican la naturaleza del fallo y el punto en que se ha cometido. La identificación y corrección de estos errores es, en la mayor parte de los casos, una labor bastante simple. Más preocupantes son los errores que aparecen tras haber eliminado los de sintaxis, cuando el programa ya ha entrado en funcionamiento. Son errores de esta clase la interrupción prematura del programa, la repetición interminable de un bucle, la obtención de resultados manifiestamente incorrectos o el deterioro de los datos almacenados en los archivos.

Comprobar si un programa satisface las especificaciones para las que fue creado es un proceso mal definido y bastante subjetivo, que conviene sea realizado por una persona diferente al autor del programa. El método más inmediato consiste en comparar el programa con sus especificaciones y verificar si las fases del proceso de diseño (refinamiento por etapas u otro) se han seguido correctamente. Es también importante comprobar si se satisfacen los objetivos de diseño.

Antes de exponer las técnicas de comprobación, merece la pena recordar algunas observaciones sobre la naturaleza de los programas hechas al principio del libro:

> En este contexto, el programa serviría para "centrar" al ordenador en la realización de una operación perfectamente definida... En consecuencia, los programas deben ser capaces de funcionar muchas veces, ser resistentes y mostrarse capaces de soportar el uso indebido accidental o deliberado.

El proceso de comprobación aspira a garantizar la satisfacción de todas las condiciones mencionadas en esta cita.

12.2

Algunas técnicas de comprobación

En las próximas secciones se expondrán algunas técnicas de comprobación de uso frecuente. Se orientan sobre todo a la determinación de la corrección de un programa o un módulo de programa bajo un espectro amplio de condiciones diferentes. Dos de ellas permiten seguir el avance de un pase del programa. La tercera parte de una perspectiva diferente, y se pregunta bajo qué condiciones funcionará el programa.

12.3

Pase de prueba

Se llama **pase de prueba** al recorrido de un programa o un módulo sentencia por sentencia, ejecutando todas las instrucciones a mano y anotando los resultados obtenidos. Es un proceso extraordinariamente lento y pesado, pero también es el que proporciona una visión más cabal del funcionamiento del programa. Suele usarse como último recurso, cuando todas las demás técnicas han fallado, pero es igualmente útil en las primeras fases de redacción y comprobación del programa.

Su limitación principal es que sólo determina que el programa funciona —o no funciona— para un conjunto concreto de datos. Es preciso adjudicar a los datos valores sencillos y los bucles sólo pueden repetirse un número limitado de veces; además, si el programa tiene gran cantidad de trayectorias lógicas, puede ocurrir que sea imposible seguirlas todas.

12.4

Ejemplo 12.1

El bucle de cálculo del programa ejemplo 11.1 servirá para hacer una demostración de la técnica del pase de prueba. La porción relevante del programa es la que aparece reproducida a continuación, sin las cláusulas **REM**. Se han escogido datos adecuados y se han hecho tres repeticiones del bucle (el programa calcula la raíz cuadrada de un número **N** con un grado de precisión **A**. Los valores usados son **N** = 16, **A** = 0,5).

Instrucciones del programa	Primera repetición	Segunda repetición	Tercera repetición
165 LET S = N/2	S=16/2, S=8		
180 LET S = 1/2*(S+N/S)	S = 1/2 × (8 + 16/8) = 5	S=1/2×(5+16/5) = 4.1	S=1/2×(4.1+16/4.1) = 4.0012
195 IF ABS(N-S*S)> ABS(A) THEN 180	ABS(N-S×S)=9	ABS(N-S×S)=.81	ABS(N-S×S)=.01
	ABS(A) =0.5 Condición cierta	ABS(A) =0.5 Condición cierta	ABS(A) =0.5 Condición falsa
205 PRINT "RAIZ CUADRADA:"; S			Valor a la salida = 4.0012

El pase de prueba revela claramente el funcionamiento del segmento de programa para los valores escogidos. El valor obtenido para la raíz cuadrada está dentro del grado de exactitud deseado.

El pase de prueba ha demostrado que el segmento de programa funciona para los valores elegidos, a saber, **N** = 16, **A** = 0,5. Aunque puede afirmarse con bastante seguridad que funcionará para otros valores semejantes, no es posible extraer ninguna conclusión sobre las condiciones generales bajo las que el segmento de programa funcionará —o no— correctamente.

12.5 Datos de prueba

Una técnica muy potente para determinar la corrección de un programa o un módulo de programa es poner éste en acción con una serie de **datos de prueba** elegidos con cuidado. Este proceso permite probar el programa bajo circunstancias muy variadas. Lo más importante de esta técnica de comprobación es la elección de los datos de prueba, extremo que examinaremos en los próximos párrafos.

En primer lugar, hay que incluir algunos valores muy sencillos, que produzcan resultados susceptibles de ser verificados manualmente. De esta forma se comprueba si el programa funciona en casos elementales.

Otros datos deben reflejar el uso normal del programa. Si éste incluye varias trayectorias lógicas, es importante que los diversos grupos de datos obliguen a recorrer todas.

Por último, es imprescindible incluir un grupo de datos "raros" para poner a prueba la robustez del programa y tratar de hacerlo fallar.

En unión de los datos de prueba, a veces es útil insertar también instrucciones de salida adicionales que indiquen los valores de las variables de trabajo, de los contadores de bucles, etc. Conviene igualmente incluir **puntos de interrupción** que detengan el programa

hasta el momento en que el usuario esté en condiciones de continuar con él. En Basic, los puntos de interrupción suelen adoptar la forma de solicitud de introducción de un carácter.

12.6 Ejemplo 12.2

Haremos la demostración de los datos de prueba sobre el programa ejemplo 5.5. El programa, junto con una instrucción de salida adicional y un punto de ruptura (líneas 152 y 153), aparece escrito a continuación (el programa, recuerde, sirve para calcular el número de años que necesita cierta población para duplicar su tamaño sabiendo su tasa de crecimiento anual; la instrucción adicional de salida indica el valor de la población cada año).

Programa

```
100 REM PROGRAMA EJEMPLO 5.5
105 REM CALCULOS DE POBLACION
110 REM MODIFICADO POR MOTIVOS DE COMPROBACION
115 PRINT "PORCENTAJE DE INCREMENTO?"
120 INPUT R
125 IF R > 0 THEN 140
130 PRINT "POR FAVOR INTRODUZCA UN NUMERO POSI
           TIVO";
135 GOTO 120
140 LET T = 0
145 LET T = T + 1
150 LET X = (1 + R/100) ^ T
152 PRINT "POBLACION EN EL ANO "; T ;" :"
153 PRINT T, X
154 PRINT
155 IF X < 2 THEN 145
160 PRINT "LA POBLACION SE DUPLICO DESPUES DE";
165 PRINT T; "ANOS"
170 END
```

Datos de prueba elegidos

Como datos de prueba hay que elegir diversos valores de la tasa de crecimiento **R** (porcentaje anual). Un caso sencillo inmediato es **R** = 100, que provocaría la duplicación de la población cada año. Los valores "típicos" están comprendidos dentro del intervalo **R** = 1 a **R** = 10.

Son valores "raros" los negativos, que deben ser rechazados, y el cero, que también debe ser rechazado. Los valores positivos de **R** muy pequeños, como **R** = 0,0000001, podrían llevar al programa a atascarse en un bucle interminable. En conjunto, los datos propuestos darán una idea cabal de las condiciones bajo las que el programa trabajará correctamente.

12.7

Determinación de las condiciones bajo las que funcionará el programa

Los dos métodos de prueba expuestos —el pase de prueba y los datos de prueba— determinan si el programa funciona o no con un conjunto determinado de valores. Pero el problema puede también abordarse a la inversa: ¿bajo qué condiciones funcionarán un programa o un segmento de programa?

La pregunta admite una respuesta rigurosa, que demuestre la corrección del programa como si de un teorema matemático se tratase, aunque esta técnica está muy lejos del alcance de este libro. Aquí trataremos de responder a la pregunta de un modo menos formalizado.

Lo que haremos será examinar con atención un programa o, preferiblemente, un módulo de un programa y determinar para qué intervalo de valores de datos funcionará correctamente. Si los módulos están interrelacionados y pasan valores de datos de unos a otros, hay que determinar también el intervalo de valores que es capaz de generar cada uno de ellos; si alguno de los módulos genera un intervalo de valores de datos más amplio que el aceptable por cualquiera de los que deben recibir dichos valores, es posible que se produzcan errores.

12.8

Ejemplo 12.3

Para ilustrar este método de verificación emplearemos el programa principal y el subprograma del ejemplo 9.2. Veamos primero el subprograma:

```
1000 REM SUBPROGRAMA FACTORIAL
1005 REM PARAMETROS
1010 REM N : NUMERO
1015 REM F : FACTORIAL
```

```
1020 REM
1025 ON SGN(N) + 2 GOTO 1100, 1200, 1300
1030 REM
1100 REM VALOR NEGATIVO DE N
1105 LET F = 0
1110 RETURN
1200 REM N VALE 0
1205 LET F = 1
1210 RETURN
1300 REM VALOR POSITIVO DE N
1305 LET F = 1
1310 FOR K = 1 TO N
1315 LET F = F * K
1320 NEXT K
1325 RETURN
```

El funcionamiento del subprograma se examina para varios valores del parámetro **N**:

Para valores negativos de **N**, se obtiene el valor **F** = 0.

Para valores nulos de **N**, se obtiene el valor **F** = 1.

Si **N** es un entero positivo, en la mayor parte de los casos se obtiene el valor correcto del factorial, pero, como el factorial de un número grande es otro mucho más grande, **N** tiene un límite máximo que depende del ordenador con que se trabaje; por encima de ese límite, se producirá un error por desbordamiento.

Si **N** es positivo pero no entero, el valor del factorial obtenido no será correcto.

Por tanto, existen valores de **N** para los que el subprograma fallará.

Veamos ahora el programa principal usado para probar el subprograma:

```
100 REM PROGRAMA EJEMPLO 9.2
105 REM PROGRAMA PRINCIPAL DE DEMOSTRACION
110 REM PARA EL SUBPROGRAMA FACTORIAL
115 REM
200 PRINT "NUMERO?" ;
205 INPUT I
210 IF I < 0 THEN STOP
215 LET N = I
220 GOSUB 1000 : REM SUBPROGRAMA FACTORIAL
225 LET J = F
230 PRINT "FACTORIAL" ; J
235 GOTO 200
240 REM FINAL DEL PROGRAMA PRINCIPAL
```

El programa se comprueba para determinar qué valores de **N** pasará al subprograma.

El valor de **N** depende de la variable **I** que se introduzca, y que puede tener cualquier valor, aunque los negativos provocan la interrupción del programa antes de asignar valor alguno a **N**. Por tanto, al subprograma llegan únicamente valores de **N** nulos o positivos.

Al examinar el subprograma a la luz de esta observación, se concluye que:

la combinación de programa principal y subprograma no conseguirá producir un resultado correcto si se introduce un valor de **N** positivo pero no entero y producirá un error por desbordamiento si el valor de **N** es muy grande.

Por tanto, el programa funcionará correctamente en determinadas circunstancias, pero no en todas.

12.9 Conclusión

Hemos visto en este capítulo algunos de los objetivos de la verificación de programas y tres técnicas de llevarla a cabo. Los puntos más importantes son los siguientes:

- El objetivo principal de la comprobación de un programa es determinar si funciona correctamente en todas las posibles condiciones de uso (o de abuso).
- El pase de prueba es un método de comprobación manual muy completo, pero limitado.
- La introducción de datos de prueba permite comprobar la corrección de un programa en circunstancias más variadas; resulta útil añadir instrucciones de salida y puntos de interrupción adicionales. La elección de los datos es importante.
- Una técnica muy potente de comprobación consiste en determinar las circunstancias bajo las que puede funcionar cada uno de los módulos de un programa. De esta forma puede evaluarse la corrección o incorrección del programa completo.

12 Ejercicio

1. Defina brevemente los siguientes términos: error de sintaxis, error de ejecución, pase de prueba, punto de interrupción.

2. Haga una comparación breve entre los tres métodos de comprobación de programas expuestos en este capítulo.
3. El siguiente algoritmo analiza el proceso de división del entero **B** por el entero **A**:

   Hacer el cociente **Q** = 0.
   Repetir el proceso:

      Restar **A** a **B**.
      Sumar 1 a **Q**.

   Hasta que **B** se haga negativo.
   Sumar **A** a **B**.
   Restar 1 a **Q**.

   El algoritmo produce el cociente como variable **Q** y el valor final de **B** como resto. Se ha interpretado mediante el siguiente programa:

```
1000 REM DIVISION ENTERA POR RESTAS SUCESIVAS
1005 REM
1010 LET Q = 0
1015 LET B = B - A
1020 LET Q = Q + 1
1025 IF B > 0 THEN 1015
1030 LET B = B + A
1035 LET Q = Q + 1
```

   Tal como está escrito, el programa contiene un error lógico.
   a) Haga un pase de prueba del segmento de programa dando valores a los datos para localizar el error.
   b) Corrija dicho error.
   c) Amplíe el segmento de programa hasta convertirlo en un subprograma y hágalo funcionar con un conjunto adecuado de datos de prueba. Escriba un programa principal sencillo de entrada/salida e incluya en el subprograma las instrucciones adicionales de salida y los puntos de interrupción que considere necesarios.
   d) Deduzca de los resultados obtenidos en *c*) y de las instrucciones del programa los intervalos de valores de **A** y **B** para los que funcionará correctamente el segmento indicado.
   e) Escriba, a partir de la línea 900, una serie de instrucciones adicionales para "filtrar" los valores de **A** y **B** susceptibles de provocar errores.
4. Aplique el tercer método de comprobación expuesto en este capítulo al programa ejemplo 11.2. Si es preciso, corrija el programa para que funcione con un intervalo más amplio de datos de entrada.
5. *a*) Inserte una instrucción de salida y un punto de interrupción adicionales en el programa ejemplo 4.2 para hacer que lleve a la salida el entero al azar generado y dé al usuario la oportunidad de crear un conjunto adecuado de datos de prueba.
   b) Haga varios pases de prueba con diferentes datos elegidos de forma que se recorran todas las trayectorias lógicas del programa.
6. *a*) Haga un pase de prueba del programa ejemplo 6.2 con datos del valor adecuado.
   b) Añada las instrucciones necesarias para convertir el programa en un subprograma y escriba un programa principal de entrada y salida.
   c) Verifique el programa escrito en *b*) con un conjunto adecuadamente elegido de datos de prueba.

d) Deduzca de las pruebas de *c*) y del contenido de las instrucciones el intervalo de valores de datos dentro del que el programa funcionará correctamente.

e) Incluya un subprograma adicional de validación para rechazar los datos susceptibles de provocar errores.

7. Aplique alguna de las técnicas de comprobación estudiadas en este capítulo a los programas que haya escrito y, si es necesario, modifíquelos a la luz de los resultados.


RESET
RETURN

13 Documentación de programas

Este capítulo se refiere al material escrito que siempre se produce en conjunción con un programa, material que se conoce como **documentación del programa.** Se expondrán los diversos tipos de documentos que pueden crearse junto con sus objetivos y algunas de las técnicas usadas para escribirlos; además, se dará un ejemplo de cada tipo de documento.

Es un capítulo estrechamente relacionado con todos los de esta parte del libro, que en conjunto se refieren a los diversos aspectos de la redacción de programas. Además, lo expuesto aquí será aplicable a todos los programas, puesto que todos necesitan documentación.

Como en los demás capítulos de esta parte, el material analizado es independiente de cualquier lenguaje de programación, y en realidad se encuentra bastante más cerca de otro lenguaje habitualmente olvidado en los círculos relacionados con la informática: el español.

13.1 Tipos de documentación

Es importante diferenciar claramente entre los distintos tipos de documentos encargados de describir un programa, porque los objetivos de los mismos son muy diferentes.

Por un lado está la **documentación para el programador** que describe, desde un punto de vista técnico, el funcionamiento del programa. Por otra parte está la **documentación para el usuario,** que describe cómo hay que utilizar el programa. Estas dos son las áreas de documentación más importantes, y se describirán en las próximas secciones de este capítulo.

A veces se crean documentos de otro tipo: así, la **documentación para el operador** informa a éste sobre cómo cargar un programa determinado. Esta clase de documentación está fuera del alcance de este libro.

13.2
Documentación para el programador

La documentación para el programador es la descripción técnica del trabajo de un programa. Va dirigida a programadores, analistas de sistemas, directores de centros de cálculo y otros profesionales de la informática. En la próxima sección se recogen los objetivos que debe satisfacer.

13.3
Objetivos de la documentación para el programador

Los más importantes son los siguientes:

- Proporcionar una descripción cabal de la función, la estructura y el método de trabajo de cada uno de los módulos del programa.
- Describir la organización del programa a partir de sus módulos componentes.
- En la industria informática, la mayor parte de los grandes programas se escriben en equipo, y la documentación para el programador permite a los diversos miembros de un equipo entender el funcionamiento de los módulos escritos por sus colegas.
- Aportar a cualquier programador que no haya participado en la creación del programa original información suficiente para que pueda entenderlo y modificarlo.
- Contribuir a la evaluación de un programa y a la determinación de si satisface las especificaciones para las que se creó.

13.4

Técnicas de redacción de la documentación para el programador

Redactar un documento que satisfaga los objetivos expuestos en la sección anterior no es tarea fácil, y aquí daremos algunos consejos para llevarla a cabo. Estas técnicas no son reglas rígidas, sino normas generales y sugerencias.

- Tan importante como la estructura general de un programa es la de la documentación. En muchos casos, ambas estructuras están relacionadas. Antes de empezar a redactar un documento, conviene dedicar un tiempo a planificar su estructura.
- Al estructurar un documento, es aconsejable separar la descripción de la función desempeñada por un programa o un módulo de un programa de la correspondiente a su forma de trabajar. En otras palabras: hay que separar "qué hace" de "cómo lo hace".
- Aunque la documentación para el programador debe ser completa, el exceso de detalle llega a ser desconcertante. Conviene recordar que el máximo nivel de detalle corresponde a las propias instrucciones del programa. Un programa bien diseñado y escrito en un lenguaje de alto nivel, como el Basic, constituye en cierto modo **su propia documentación.** Dicho de otra forma: el programa describe su estructura y su método de trabajo.
- Esta documentación va dirigida a una audiencia experta y, por tanto, no hay inconveniente en usar términos técnicos de informática. Sin embargo, en muchos casos, el uso de términos técnicos sirve simplemente para encubrir la extraordinaria pobreza del español en que está redactado el documento. Es preciso insistir en que se trata de un recurso nada aconsejable y en que es imprescindible acostumbrarse a redactar la documentación en un estilo claro y conciso.

Estas normas pueden resumirse como sigue:

Estructurar la documentación con cuidado, separando sobre todo el "qué hace" del "cómo lo hace".
Ser breve.
Utilizar un estilo claro y conciso.

13.5

Variables y diagramas de flujo

La documentación para el programador debe incluir la definición de todas las variables que forman parte del programa. Respecto a los diagramas de flujo, no hay acuerdo. Hay tres opciones básicas: diagramas de flujo sumarios que resuman la corriente general de control del programa; diagramas de flujo detallados que respondan con cierta fidelidad a las instrucciones del programa; y renuncia a los diagramas de flujo. En este libro se ha optado por la segunda solución.

13.6

Ejemplo 13.1

Examinaremos a continuación la documentación para el programador correspondiente al programa ejemplo 11.1.

Programa ejemplo 11.1: cálculo de la raíz cuadrada

Función

El programa calcula la raíz cuadrada de un número con un grado de exactitud dado.

Estructura

El programa tiene tres segmentos —entrada y validación, proceso y salida— que se describen a continuación.

Segmentos de entrada y validación

Este segmento acepta un número y un grado de precisión. Si el primero es negativo, lo rechaza y solicita al usuario la introducción de otro positivo. El grado de precisión es la diferencia máxima entre el número y el cuadrado de su raíz calculada por el programa.

Segmento de procesado

El programa trabaja de acuerdo con el siguiente principio:

Dada una estimación s de la raíz cuadrada de un número n, se obtiene una estimación s^+ más exacta mediante la fórmula

$$s^+ = \frac{1}{2}\left(s + \frac{n}{s}\right)$$

Esta fórmula se utiliza en el algoritmo del proceso:

Elegir una estimación inicial de la raíz cuadrada.
Repetir el proceso:
    Calcular una estimación más exacta mediante la fórmula anterior.
Hasta que la estimación tenga el grado de precisión deseado.

Segmento de salida

A la salida se obtiene la raíz cuadrada obtenida mediante el proceso que acaba de exponerse.

Observación: el método propuesto no constituye la forma más eficaz de calcular una raíz cuadrada, pero el programa produce resultados, incluso para números grandes, con una rapidez aceptable.

Variables

N Número del que quiere calcularse la raíz cuadrada.
P Grado de precisión del cálculo.
S Estimación de la raíz cuadrada.

Diagrama de flujo

Véase figura 11.1.

13.7 Documentación para el usuario

Esta información contiene las instrucciones de uso del programa. Va dirigida a los usuarios que, en la mayor parte de los casos, no son especialistas en ordenadores.

La documentación para el usuario es mucho más difícil de redactar que la dirigida al programador. Hay que luchar continuamente contra la tentación de usar un lenguaje técnico, adecuado para un especialista en informática, pero no necesariamente para el usuario. En la próxima sección veremos los objetivos que debe satisfacer esta documentación.

13.8

Objetivos de la documentación para el usuario

La forma que adopte esta documentación depende en gran medida del tipo de programa y del tipo de usuario, y por ello es difícil fijar unos objetivos de validez universal. Los que a continuación se relacionan deben considerarse como un mero punto de partida:

- Informar al usuario de las posibilidades del programa. En otras palabras: ponerle al tanto de lo que puede esperarse del programa.
- Proporcionar instrucciones detalladas paso a paso sobre el uso del programa.
- Explicar el significado de todos los mensajes que pudieran aparecer, particularmente de los de error.
- Señalar cómo se solucionan los errores.
- Explicar todos los términos técnicos que pudieran aparecer durante el uso del programa.
- Tener un nivel de redacción accesible a todos los posibles usuarios del programa.

13.9

Técnicas de redacción de la documentación para el usuario

Como en el caso anterior, las indicaciones contenidas en esta sección deben considerarse normas y sugerencias generales, no reglas estrictas.

- Averiguar los conocimientos, posibilidades y limitaciones del posible usuario. Para ello hay que tratar de determinar el nivel de comprensión de documentos escritos y los conocimientos matemáticos e informáticos del usuario medio. Este aspecto es obvio, además de imprescindible, pero se descuida con mucha frecuencia. No siempre es fácil reunir la información necesaria.

- La redacción ha de hacerse desde la perspectiva del usuario, teniendo siempre en cuenta el perfil del mismo elaborado en el paso anterior.
- Estructurar la documentación con cuidado, separando "lo que puede hacer" de "cómo lo hace".
- Describir los pasos del programa uno por uno, dedicando especial atención a los puntos de bifurcación.
- Incluir ejemplos prácticos en los casos en que sea necesario.
- Explicar todos los mensajes e indicaciones contenidos en el programa.
- Indicar con toda claridad la forma de solucionar los errores.
- Evitar la jerga técnica y las explicaciones prolijas sobre el funcionamiento del programa.
- Utilizar un estilo claro, conciso y accesible al usuario medio.
- Ser breve.

Todas estas sugerencias pueden resumirse como sigue:

> Determinar el perfil del usuario medio y adaptar al mismo la redacción del documento.
>
> Estructurar con cuidado el documento y describir los pasos sucesivos uno por uno.
>
> Explicar el contenido de todos los mensajes e indicaciones y adjuntar instrucciones para resolver los errores.
>
> Incluir ejemplos prácticos.

13.10 Ejemplo 13.2

Esta sección contiene las instrucciones para el usuario del programa ejemplo 9.1. El programa en cuestión sirve para emplear un ordenador como una sencilla calculadora. Se supone que los usuarios serán alumnos de un colegio con una edad media de 10 años.

Para usar el ordenador como una calculadora

Este programa te permitirá utilizar un ordenador como una simple calculadora, con la que podrás hacer operaciones sencillas, como:

23 + 47 =

o cálculos más largos como

13,6 × 41,2 × 1,35 ÷ 9,4 =

Números y operaciones

Para usar el programa y las **operaciones** por medio del teclado, tienes que introducir los **números,** como 15 ó 3,54, + para sumar, − para restar, * para multiplicar, / para dividir y = para calcular el resultado.

Realización de un cálculo sencillo

Para ejecutar un cálculo sencillo como

23 + 17 =

- teclea el primer número,
- pulsa el signo de la operación,
- teclea el segundo número,
- pulsa el signo =

La solución aparecerá en la pantalla.

Ejemplo: calcular

23 + 17 =

- teclea 23,
- pulsa +, Verás el número 23
- teclea 17,
- pulsa = Verás el número 40, que es la solución.

En caso de error

Si pulsas una tecla que no corresponde a ningún signo de operación, aparecerá en la pantalla el mensaje **OPERACION EQUIVOCADA**. Pulsa a continuación la tecla correcta.

Si no pulsas un número correcto, aparecerá en la pantalla el signo de interrogación ? Vuelve a pulsar el número sin equivocarte.

Realización de cálculos largos

(Esta parte de la documentación queda como ejercicio.)

13.11

Conclusión

Hemos estudiado en este capítulo los dos tipos de documentos más frecuentes, dirigidos, respectivamente, al programador y al usuario. Los puntos de más interés son los siguientes:

- La documentación para el programador describe, a nivel especializado, el funcionamiento del programa, su estructura, la función que cumple y la forma en que trabaja. Uno de sus principales objetivos es permitir a cualquier programador comprender y modificar el programa, aunque no haya participado en su elaboración.
- La documentación para el usuario contiene las instrucciones de empleo del programa. Recoge sus posibilidades y expone paso a paso su utilización. De particular importancia son las instrucciones de resolución de errores. Esta documentación no es de orden técnico, y ha de estar adaptada a la capacidad del usuario medio al que se dirige.

13

Ejercicio

1. Defina brevemente los siguientes términos: documentación para el programador, documentación para el usuario, programa autodocumentado.
2. ¿Cuáles son las diferencias principales entre la documentación para el programador y la documentación para el usuario?
3. Complete la guía para el usuario del programa ejemplo 9.1 iniciada en el ejemplo 13.2.
4. Redacte la documentación para el programador, correspondiente al módulo de servicio **BUSCAR** del programa ejemplo 11.2. Se sugiere usar los siguientes encabezamientos:

   Programa ejemplo 11.2; módulo de servicio **BUSCAR**.
   Función.
   Estructura.
   Forma de funcionamiento.

5. Escriba una guía para el usuario de los dos programas ejemplo del capítulo 10 (y para el programa escrito como respuesta a la pregunta 4 del ejercicio 10, si es que lo ha realizado). Todos estos programas se refieren a la creación de una agenda de nombres, direcciones y números de teléfono. Suponga que los usuarios son adultos.
6. Escriba la documentación para el programador correspondiente al subprograma del ejemplo 9.2.
7. Escriba la documentación para el programador (o para el usuario) correspondiente a los programas que haya realizado usted mismo. En el segundo caso, indique las características de los supuestos usuarios.


RESET
RETURN

14
Mantenimiento de programas

Al principio de este libro, hablábamos de un programa como un medio de "centrar" un ordenador en la realización de una tarea específica, y recordamos a propósito de esa descripción que muchos programas se diseñan para ser usados de forma continua o repetida.

Por varias razones, que expondremos a continuación, los programas sometidos a uso continuo o repetido deben corregirse de vez en cuando; este proceso de corrección o puesta al día es lo que se llama **mantenimiento del programa.**

Las razones a que hacíamos referencia son, entre otras:

- La experiencia en el uso de un programa lleva a detectar defectos o errores que habían pasado desapercibidos.
- La tarea que realiza el programa sufre también con el tiempo cambios en sus características.
- El ordenador en que se carga el programa puede cambiarse por un equipo más moderno.

Veremos en este capítulo las modalidades de mantenimiento de programas que es preciso conocer y daremos algunas normas generales sobre la forma de llevar a cabo dicho mantenimiento. La discusión de cada uno de los tipos de mantenimiento va acompañada de un ejemplo resuelto. El capítulo concluye con una sección de mantenimiento y diseño de programas.

Este es el último capítulo del libro en el que los programas se estudian desde un punto de vista muy general, independiente de cualquier lenguaje de programación particular. La finalidad de estos cuatro capítulos de orden general es crear unos cimientos firmes sobre los que basar el acercamiento a los capítulos restantes, en los que vuelve a hacerse uso de los elementos del Basic presentados con anterioridad.

14.1

Tipos de mantenimiento de programas

Las labores de mantenimiento de programas se dividen en tres categorías amplias según el grado de modificación necesario:

- modificación de un módulo del programa;
- sustitución de un módulo del programa por otro nuevo;
- modificación de la estructura del programa, con inserción de módulos adicionales, eliminación de otros o modificación de las relaciones entre ellos.

Estos tipos no se consideran divisiones rígidas, y la actualización de un programa puede incluir en la práctica labores de mantenimiento de varios tipos. Sin embargo, la división expuesta constituye un punto de partida útil para el estudio de las técnicas de mantenimiento. Se discutirán más adelante en secciones separadas.

14.2

Algunas normas generales

Antes de examinar las técnicas específicas de modificación de programas, será útil hacer algunas observaciones generales, que no deben considerarse como reglas inflexibles, sino como sugerencias interesantes.

Al abordar la corrección de un programa, hay que tener en cuenta los siguientes puntos:

- Establecer los objetivos del cambio. En concreto, es importante distinguir entre cambios para adaptar el programa a las especificaciones de partida y cambios en las propias especificaciones.
- Determinar el nivel de actualización necesario. Para ello se identifica el tipo (o los tipos) de cambios descritos en la sección anterior.

- Estar atento para no introducir errores en el programa durante el proceso de actualización.
- No enredar en los programas que funcionan correctamente. Nunca se insistirá lo suficiente en la importancia de este punto. Las modificaciones innecesarias siempre producen más daños que beneficios.

Pasaremos ahora a discutir cada uno de los tipos de mantenimiento expuestos antes a la luz de estas consideraciones generales.

14.3 Modificaciones dentro de los módulos del programa

Son cambios a escala relativamente reducida, necesarios cuando es preciso modificar una función concreta del programa. Los cambios posibles de este tipo son muy variados: modificación del número de veces que debe repetirse un bucle, inserción de un segmento de programa en un bucle, alteración de los valores iniciales de las variables o cambios en la estructura de la entrada o la salida.

Al introducir esta clase de modificaciones, lo más importante es evitar los errores. La comprobación a fondo del módulo revisado es imprescindible.

14.4 Ejemplo 14.1

Aunque el ejemplo 3.3 es un programa completo, sus reducidas dimensiones permiten también considerarlo como un módulo de un programa más amplio. El programa es un sistema rápido de cálculo de interés compuesto que realiza un solo cálculo en cada pase. Se trata de modificarlo de manera tal que pueda utilizarse repetidamente.

Método

El programa puede describirse mediante el algoritmo siguiente:

Visualizar encabezamiento.
Introducir los datos necesarios.
Realizar el cálculo del interés compuesto.
Mostrar el resultado.

La estructura más adecuada al programa modificado es la de un bucle "repetir hasta que":

Visualizar encabezamiento.
Repetir el proceso:

Introducir los datos necesarios.
Realizar el cálculo del interés compuesto.
Mostrar el resultado.
Preguntar al usuario si hay que volver a ejecutar el cálculo.

Hasta que no se responda **SI**.

Variable adicional

W$ Respuesta del usuario a la pregunta sobre la repetición.

Diagrama de flujo

La figura 14.1 ilustra el flujo de control del programa modificado.

Programa modificado

```
100 REM PROGRAMA EJEMPLO 3.3
105 REM CALCULOS DE INTERES COMPUESTO
110 REM MODIFICADO PARA REALIZAR CALCULOS REPETI
        TIVOS
115 PRINT "CALCULOS DE INTERES COMPUESTO"
120 PRINT
125 PRINT
130 PRINT "CANTIDAD AHORRADA (PESETAS)";
135 INPUT P
140 PRINT "INTERES";
145 INPUT I
150 PRINT "TIEMPO DE IMPOSICION";
155 INPUT T
160 PRINT
165 LET A = P*(1 + I/100)^T
170 PRINT "DINERO ACUMULADO  :";A; " PESETAS"
175 PRINT
177 PRINT "DESEA REALIZAR OTRO CALCULO?";
178 INPUT W$
179 IF W$ = "SI" THEN 120
180 END
```

Puntos de interés

- El nombre de la variable adicional **W$** es diferente a todos los usados en el resto del programa.

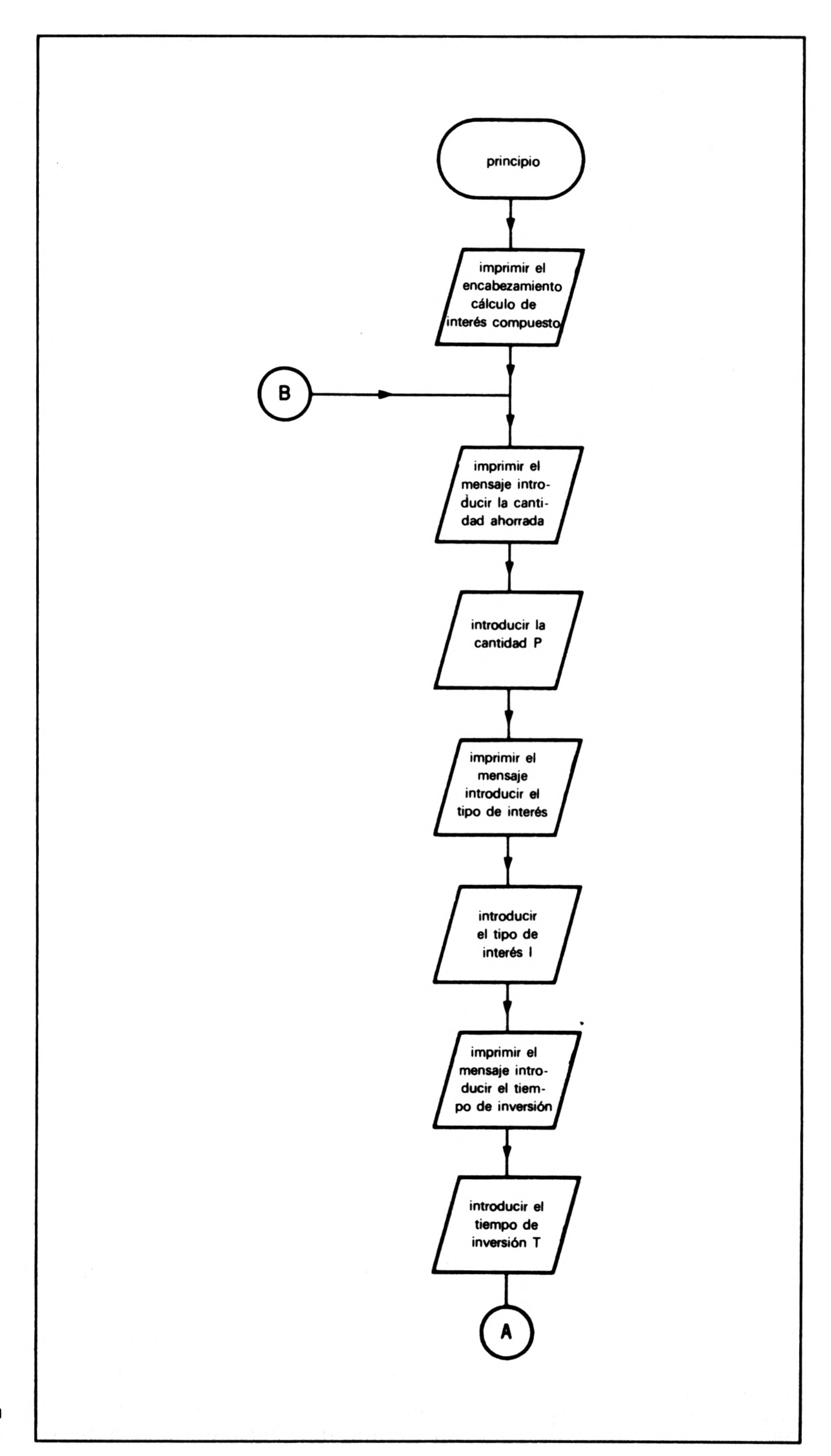


Figura 14.1
Diagrama de flujo del programa ejemplo 14.1

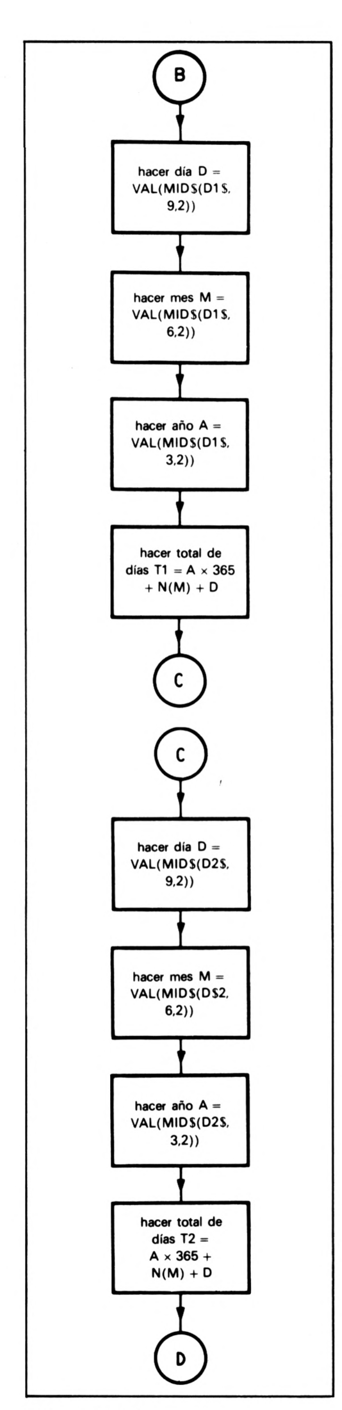


Figura 14.2
Diagrama de flujo del ejemplo 14.2

A

calcular la cantidad acumulada $A = P \times (1 + I/100)^T$

imprimir la cantidad acumulada A

imprimir la pregunta ¿desea realizar otro cálculo?

introducir la respuesta W$

¿es W$ = "sí"?

sí

B

no

fin

- Las instrucciones nuevas se introducen en el hueco que queda entre las líneas números 175 y 180 del programa original.
- La condición "repetir hasta que" aparece invertida en el programa.

14.5

Sustitución de módulos

Si una de las funciones realizadas por el programa experimenta un cambio notable o si se adopta una nueva técnica de escritura para un módulo, la forma más eficaz de modificar el programa es "desconectar" el módulo afectado y cambiarlo por otro.

Cuando se cambia un módulo, es importante conservar el mismo conjunto de relaciones —o **interfaz**— con el resto del programa que mantenía el módulo antiguo. Ello obliga a transferir los mismos datos mediante los mismos parámetros y a tener mucho cuidado para no introducir efectos secundarios.

14.6
Ejemplo 14.2

El programa ejemplo 6.2 calcula la diferencia aproximada entre dos fechas. En su forma actual acepta fechas en la forma día/mes/año, y desea modificarse para que lo haga en la forma año - mes - día.

La inspección del programa indica que es preciso cambiar los dos módulos que calculan el número de días transcurridos desde el cambio de siglo hasta la fecha dada. Su relación con el resto del programa es la siguiente: cada uno acepta una fecha (**D1$** y **D2$**, respectivamente); utilizan ambos una tabla (matriz **N**) con el número de días transcurridos hasta el principio de cada mes. Los nuevos módulos han de utilizar las mismas variables para transferir datos al resto del programa y recibirlos del mismo.

Variables

D1$ Fecha 1.
D2$ Fecha 2.
D Día.
M Mes.
A Año.
T1 Días totales desde el principio de siglo para la fecha 1.
T2 Días totales desde el principio de siglo para la fecha 2.

Diagrama de flujo

La figura 14.2 ilustra el flujo de control de los módulos del programa revisados.

Módulos del programa

Los módulos revisados quedan como sigue:

```
6200 REM SE PROCESA LA PRIMERA FECHA
6202 REM VERSION REVISADA
6205 LET D = VAL(MID$(D1$, 9, 2))
```

```
6210 LET M = VAL(MID$(D1$, 6, 2))
6215 LET A = VAL(MID$(D1$, 3, 2))
6220 LET T1= A*365 + N(M) + D
6225 REM
6300 REM SE PROCESA LA SEGUNDA FECHA
6302 REM VERSION REVISADA
6305 LET D = VAL(MID$(D2$, 9, 2))
6310 LET M = VAL(MID$(D2$, 6, 2))
6315 LET A = VAL(MID$(D2$, 3, 2))
6320 LET T2 = A*365 + N(M) + D
6325 REM
```

Puntos de interés

- Los cambios introducidos son completamente autónomos y no exigen ninguna modificación del resto del programa.
- Esta clase de cambios sólo son posibles en programas cuidadosamente estructurados en módulos.

14.7

Modificaciones de la estructura del programa

Las modificaciones más inclinadas a generar errores son las que afectan a la estructura del programa, necesarias cuando éste debe sufrir cambios de consideración.

Para hacer modificaciones estructurales es preciso repetir todo el proceso de diseño del programa, única manera de garantizar que las correcciones serán coherentes y que el programa corregido conservará una estructura sin fallos. Es preciso comprobar si se han introducido todos los cambios necesarios y asegurarse de que no se han cometido errores ni se han creado efectos secundarios.

14.8

Ejemplo 14.3

El programa ejemplo 7.2 contiene los principios de un paquete de manipulación de matrices de tipo general. El objetivo de la modificación propuesta es ampliar el alcance del programa para que ejecute las siguientes operaciones:

INT B Introducir la matriz **B**.
MPC Multiplicar la matriz **A** por una constante y dar el resultado en la matriz **C**.

Método

El programa está estructurado en varios módulos, uno de control y otro más por cada una de las operaciones. El de control debe modificarse para que tenga en cuenta las órdenes adicionales y transfiera el mando a los dos nuevos que habrán de redactarse para ejecutar las operaciones propuestas. El módulo de inicialización y los correspondientes a las operaciones no se ven afectados por esta modificación.

En el programa original se dejaron huecos suficientes en la numeración para acomodar las nuevas instrucciones del módulo de control y los módulos de operaciones adicionales.

Los nuevos módulos de operaciones tienen la misma estructura de los antiguos y utilizan como ellos bucles anidados para recorrer las filas y columnas de las matrices. Las instrucciones añadidas al módulo de control siguen igualmente el patrón de las antiguas.

Variables adicionales

K Constante por la que se multiplican todos los elementos de una matriz.

Diagrama de flujo

La figura 14.3 ilustra el flujo de control del módulo de mando modificado y del módulo de multiplicación por una constante. El módulo de introducción de la matriz **B** es igual al de introducción de la matriz **A**, ilustrado en la figura 7.2.

Módulos del programa modificados

```
2000 REM MODULO DE CONTROL MODIFICADO
2005 PRINT "ORDEN?";
2010 INPUT O$
2015 IF O$ = "INT A" THEN 2500
2020 IF O$ = "INT B" THEN 3000
2025 IF O$ = "SUM" THEN 3500
2035 IF O$ = "MPC" THEN 4500
2060 IF O$ = "SAL" THEN 7000
2065 IF O$ = "FIN" THEN 2080
2070 PRINT "ORDEN NO RECONOCIDA. POR FAVOR REPITA
          ";
2075 GOTO 2010
2080 PRINT "SE HA LLEGADO AL FINAL DEL PROGRAMA"
2085 STOP
```

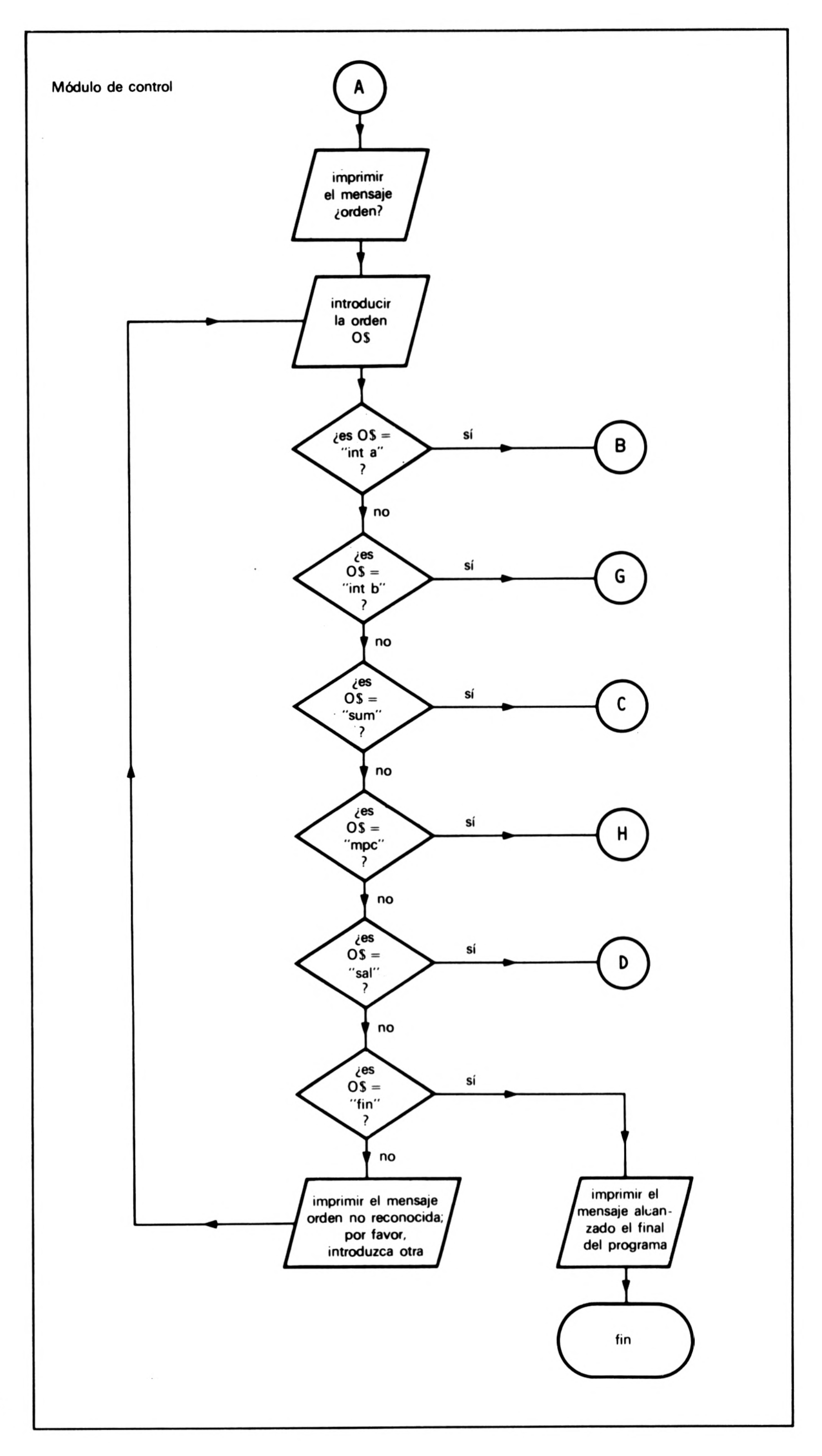
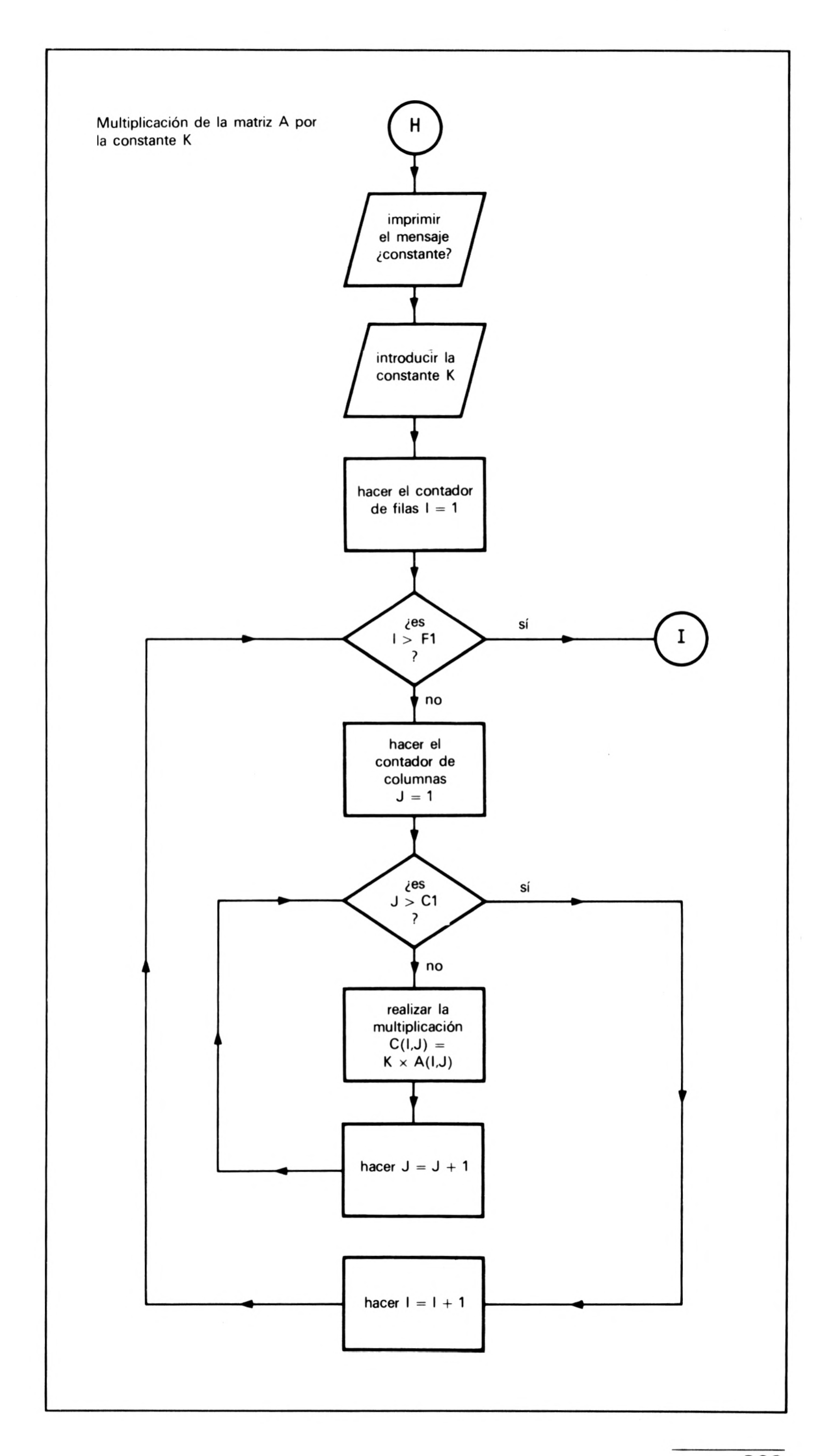


Figura 14.3
Diagrama de flujo del ejemplo 14.3


Multiplicación de la matriz A por la constante K
H
imprimir
el mensaje
¿constante?
introducir la
constante K
hacer el contador
de filas I = 1
¿es
I > F1
?
sí
I
no
hacer el
contador de
columnas
J = 1
¿es
J > C1
?
sí
no
realizar la
multiplicación
C(I,J) =
K × A(I,J)
hacer J = J + 1
hacer I = I + 1

multiplicación de la matriz A
por la constante K
(continuación)

I

hacer el tamaño de la matriz C igual al de la matriz A
R3 = R1 / C3 = C1

A

```
3000 REM ENTRADA DE LA MATRIZ B
3005 PRINT "NUMERO DE FILAS?";
3010 INPUT R2
3015 IF R2 > 0 AND R2 < 11 AND INT(R2) = R2 THEN
     3055
3020 PRINT "NUMERO INCORRECTO. POR FAVOR VUELVA A
           INTRODUCIRLO"
3025 GOTO 3010
3030 PRINT "NUMERO DE COLUMNAS?";
3035 INPUT C2
3040 IF C2 > 0 AND C2 < 11 AND INT(C2) = C2 THEN
     3055
3045 PRINT "NUMERO INCORRECTO. POR FAVOR VUELVA A
           INTRODUCIRLO"
3050 GOTO 3035
3055 FOR I = 1 TO R2
3060 PRINT "INTRODUZCA UNA FILA DE LA MATRIZ"
3065 FOR J = 1 TO C2
3070 INPUT B(I, J)
3075 NEXT J
3080 NEXT I
3085 GOTO 2000
3090 REM
4500 REM MULTIPLICACION DE UNA MATRIZ POR UNA
         CONSTANTE
4505 PRINT "CONSTANTE?";
4510 INPUT K
4515 FOR I = 1 TO R1
4520 FOR J = 1 TO C1
4525 LET C(I, J) = K * A(I,J)
4530 NEXT J
4535 NEXT I
4540 LET R3 = R1
4545 LET C3 = C1
4550 GOTO 2000
4555 REM
```

Puntos de interés

- Aunque se ha ampliado el programa, el resultante conserva la misma estructura general.
- Los nuevos módulos de operaciones siguen la misma pauta de los antiguos y, en concreto, devuelven el control a la línea 2000.
- En los módulos nuevos se utilizan los mismos contadores **I** y **J** que en los antiguos.

14.9 Mantenimiento y diseño de programas

Las técnicas de modificación de programas y los problemas que pueden presentarse durante su aplicación subrayan la importancia del buen diseño. El logro de una estructura clara, la evitación de efectos secundarios en los módulos y la documentación adecuada son las tres áreas más importantes.

Si un programa está bien estructurado y preparado, a ser posible, para aceptar modificaciones ulteriores, la alteración del mismo será casi siempre una labor sencilla. Por el contrario, si no está claramente estructurado en módulos, resultará imposible la sustitución de éstos. Además, si la estructura no está clara, las modificaciones estructurales serán difíciles de introducir.

En Basic, es importante dejar huecos en la numeración para prever ampliaciones. No hay tarea más aburrida y más expuesta a errores que el cambio de numeración de las líneas de un programa.

Si los módulos de un programa provocan efectos secundarios, la sustitución de los mismos puede provocar consecuencias inesperadas y, probablemente, irá acompañada de errores. Ni que decir tiene que los módulos nuevos no deben tampoco generar efectos secundarios.

La documentación es imprescindible en el momento de modificar un programa, sobre todo si la modificación ha de afectar a la estructura. También es importante documentar todas las modificaciones.

14.10 Conclusión

En este capítulo se han discutido las circunstancias que obligan a modificar los programas y se han presentado también algunas técnicas de modificación. Los puntos más interesantes son:

- Las razones que llevan a modificar un programa son: defectos descubiertos con la práctica, alteración de la tarea que debe ejecutar el programa o introducción de un equipo de proceso nuevo.
- Pueden efectuarse simultáneamente hasta tres tipos diferentes de modificaciones, según el alcance de las mismas: modificaciones internas a los módulos, sustitución de módulos y cambios en la estructura del programa.
- Al modificar un programa, es importante determinar los objetivos del cambio, determinar el alcance del mismo, evitar la introducción de errores y no tocar un programa si funciona correctamente.
- Las dificultades que entraña la modificación de programas subrayan la importancia del buen diseño y la documentación adecuada.

14

Ejercicio

1. Defina brevemente los siguientes términos: mantenimiento de programas, interfaz entre módulos de un programa.

2. Este segmento de programa acepta la introducción de diez números y calcula su media:

```
100 LET T = 0
105 FOR K = 1 TO 10
110 INPUT N
115 LET T = T + N
120 NEXT K
125 LET A = T/10
```

Realice las modificaciones siguientes:

a) Pase a 20 la cantidad de números.

b) Pase la cantidad de números a una variable introducida al principio del programa.

c) Cambie el programa en el sentido de finalizar la introducción de datos mediante un indicador de fin de datos.

3. El programa ejemplo 3.2 solicita la introducción de un nombre, una fecha de nacimiento y una dirección. Realice las modificaciones indicadas a continuación (tenga en cuenta que se trata de modificaciones acumulativas, porque cada una opera sobre la anterior, no sobre el programa original):

a) Cambie el programa para introducir primero el nombre y a continuación el apellido.

b) Vuelva a modificarlo para que el usuario tenga la oportunidad de volver a introducir todos los datos si aprecia algún error cuando éstos aparezcan en la pantalla.

c) Haga una nueva modificación para poder introducir un número de conjuntos de datos. Una vez que cada uno de los conjuntos haya sido

introducido, presentado en pantalla y, en caso necesario, corregido, se pregunta al usuario si desea introducir alguno más.

d) Cree un fichero e introduzca un conjunto de datos corregidos en cada uno de sus registros.

4. El programa ejemplo 4.1 es el segmento de control de un programa de control de existencias. Modifíquelo para incluir una variable de código de orden que tome el valor 1 si la orden es **RECIBIR**, 2 si es **DESPACHAR**, etcétera. Utilice a continuación una instrucción **ON ... GOSUB** para transferir el control a los módulos de operación situados en los mismos números de línea que antes.

5. El programa ejemplo 4.2 es un sencillo juego de adivinación de números para niños. Introduzca las siguientes modificaciones acumulativas:

a) Permita que el usuario introduzca un número cualquiera para lanzar el generador de números aleatorios. Utilice la técnica discutida en las secciones 8.8 y 8.9.

b) Dé al usuario la opción de utilizar el programa repetidamente, sin volver a iniciar el generador de números aleatorios tras cada repetición.

c) Mantenga una puntuación para cada pase del programa. Esta puntuación cuenta el número de conjeturas necesarias para adivinar cada uno de los números. Tras cada partida aparecen en pantalla la puntuación y el número de partidas jugadas. También se calcula y se lleva a pantalla la puntuación media por partida.

6. *a*) Modifique los programas de los ejemplos 10.1 y 10.2 para adaptarlos a los siguientes cambios introducidos en la estructura de la agenda telefónica personal:

Los nombres se introducen y almacenan en forma de dos variables, un nombre y un apellido.
Las direcciones deben constar de tres líneas más un código postal separado.
Los números de teléfono han de ampliarse para incluir un prefijo.

b) Modifique el programa de creación del fichero para que los registros puedan comprobarse y editarse antes de escribirse en aquél.

7. Modifique los módulos de servicio del programa ejemplo 11.2 de manera que no queden huecos en la matriz que almacena las identificaciones de los aparatos. Si se extrae un elemento de la matriz, los situados detrás "avanzan" para cubrir el espacio vacío. Los elementos nuevos se introducen por la parte posterior de la matriz. La modificación propuesta no debe provocar ninguna alteración en los módulos de nivel superior.


RESET
RETURN

15
Ordenación

Este capítulo es el primero de una serie de tres dedicados a otras tantas operaciones de programación fundamentales —**ordenación, búsqueda** y **refundición**— con aplicaciones muy variadas. Es importante aprender a programarlas correctamente y con eficacia.

Este es también el primer capítulo en hacer uso de las últimas técnicas aprendidas: diseño de programas, verificación, documentación y mantenimiento, por lo que está íntimamente relacionado con los cuatro precedentes.

La ordenación se estudiará primero en términos generales, junto con varias técnicas para llevarla a cabo y un programa diseñado para ejecutar una de tales técnicas de uso común. Se discutirá también el rendimiento de dicho programa, sobre el que se harán algunas modificaciones. Los ejercicios del final del capítulo darán la oportunidad de diseñar un programa de ordenación según un método diferente.

15.1
Concepto de ordenación

La idea de **ordenación** es muy sencilla: se trata de poner en orden una serie de objetos. Estos pueden ser números, que deben ordenarse

por su magnitud, o series de caracteres, que deben ordenarse alfabéticamente.

Pero en la mayor parte de las situaciones de cálculo, la ordenación puede convertirse en una operación sorprendentemente difícil. Unas veces el problema está en que el conjunto de datos es demasiado grande para la memoria del ordenador y otras ocurre que los objetos a ordenar no son datos individuales, sino registros. Estos se identifican mediante uno o más datos llamados **claves**, que determinan el orden de las fichas.

Aquí evitaremos deliberadamente estos problemas y estudiaremos la ordenación en relación con grupos de datos pequeños formados por números o palabras. Sin embargo, las técnicas presentadas pueden ampliarse y modificarse para resolver con ellas los problemas mencionados en el párrafo anterior. Tales ampliaciones y modificaciones están fuera del alcance del presente libro, que se limita a unas pocas técnicas de ordenación esenciales.

15.2
Técnicas de ordenación

Se han desarrollado varias técnicas para ordenar datos con ordenador sobre las que se ha investigado exhaustivamente, particularmente desde el punto de vista de la eficacia, porque la ordenación es a veces un proceso muy lento y hay gran demanda de procedimientos de ordenación rápidos.

En la mayor parte de las versiones del Basic, las técnicas disponibles se ven limitadas por la imposibilidad de escribir programas recurrentes, lo que obliga a prescindir de algunas de las más eficaces, como la "ordenación rápida". De todas formas, las dos expuestas en este capítulo —la **ordenación por el método burbuja** y la **ordenación por selección**— se adaptan bien al Basic y son razonablemente eficaces.

Hay otro procedimiento muy útil para ordenar datos en una secuencia aproximada conocido por **barajado** que se expondrá en los ejercicios situados al final del capítulo.

15.3
Ordenación por el método burbuja

La técnica del **método burbuja** se llama así porque los números menores "ascienden flotando" hasta la parte superior del conjunto a medida que el proceso avanza. La técnica se aplicará aquí a datos

numéricos, pero funciona igualmente con caracteres alfabéticos.

El algoritmo del método burbuja más sencillo es el siguiente:

Repetir el proceso:

> Comparar cada uno de los pares consecutivos de números del conjunto.
> Si los números del par no están en orden, intercambiarlos.

Hasta que todas las parejas queden en orden.

Si queremos ordenar por orden creciente los números 10, 7, 3, se sigue el proceso siguiente:

10 ↔ 7 3	Comparar 10 con 7. Como no están en orden ascendente, se intercambian.
7 10 ↔ 3	Comparar 10 con 3. Como no están en orden ascendente, se intercambian. Repetir el proceso, porque no todos los pares están en orden.
7 ↔ 3 10	Comparar 7 con 3. Como no están en orden ascendente, se intercambian.
3 7 ↔ 10	Comparar 7 con 10. No se intercambian, porque están en orden. Aunque parece que el conjunto ya está ordenado, es preciso repetir el proceso una vez más para confirmarlo.

Este sencillo ejemplo da una idea de la cantidad de tiempo que consume el proceso. Más adelante, en los ejercicios, se discutirán otros medios de acelerarlo. Vamos a diseñar ahora un programa que ejecute el algoritmo expuesto, siguiendo para ello el método del refinamiento por etapas.

15.4
Programa ejemplo 15.1

La descomposición inicial más obvia de la tarea de ordenación es:

Introducir una serie de números.
Ordenarlos en orden ascendente.
Mostrar el conjunto ordenado.

La inclusión del algoritmo de ordenación lleva al primer refinamiento:

Introducir una serie de números.
Repetir el proceso:

Comparar cada uno de los pares consecutivos de números del conjunto.
Si los números del par no están en orden, intercambiarlos.

Hasta que todos los pares estén ordenados.
Mostrar el conjunto ordenado.

Antes de avanzar en el proceso de refinamiento, hay que elegir las variables y estructuras de datos adecuadas.

La **matriz** es la estructura de datos más adecuada para el conjunto de números. Si el programa ha de manejar conjuntos de longitud variable, es preciso introducir la cantidad de números de cada uno y comprobarlo frente al límite de la matriz antes de continuar el programa.

Para determinar si todos los pares de números están en orden, se lleva a cabo el siguiente proceso: antes de cada conjunto de comparaciones, se pone a cero una **variable de verificación**; si se efectúa un intercambio, la variable pasa a valer 1. Si, tras una serie de comparaciones, la variable de verificación sigue valiendo cero, es que no se han realizado intercambios, lo que significa que todos los pares están en orden.

Esto conduce al refinamiento final:

Repetir el proceso:

Introducir la cantidad de números del conjunto.

Hasta que este número no exceda del límite de la matriz.
Introducir los números del conjunto.
Repetir el proceso:

Poner a cero la variable de verificación.
Comparar cada uno de los pares consecutivos de números del conjunto.
Si los números del par no están en orden, intercambiarlos y poner a 1 la variable de verificación.

Hasta que la variable de verificación valga siempre cero.
Mostrar el conjunto ordenado de números.

Esta descomposición en pasos es suficientemente detallada como para poder escribir un programa a partir de ella.

Variables

N(25) Matriz de números.
M Cantidad de números del conjunto.
K Contador del bucle.
C Variable de verificación.
U Variable provisional utilizada durante el intercambio de números.

Diagrama de flujo

Véase figura 15.1.

Programa

```
100 REM PROGRAMA EJEMPLO 15.1
110 REM ORDENACION DE NUMEROS POR EL METODO
        ''BURBUJA''
120 DIM N(25)
130 REM
140 REM
200 REM SEGMENTO DE ENTRADA
210 PRINT "INTRODUCE EL TAMANO DEL CONJUNTO";
220 INPUT M
230 IF M <= 25 THEN 300
240 PRINT "EL LIMITE ES 25"
250 PRINT "POR FAVOR UTILICE UNA CIFRA MENOR"
260 GOTO 200
270 REM
300 PRINT "INTRODUCE LOS NUMEROS DEL CONJUNTO"
310 FOR K = 1 TO M
320 INPUT N(K)
330 NEXT K
340 REM
400 REM SEGMENTO DE ORDENACION
410 LET C = 0
420 FOR K = 1 TO M - 1
430 IF N(K) <= N(K + 1) THEN 480
440 LET U = N(K)
450 LET N(K) = N(K + 1)
460 LET N(K + 1) = U
470 LET C = 1
480 NEXT K
490 IF C <> 0 THEN 410
```

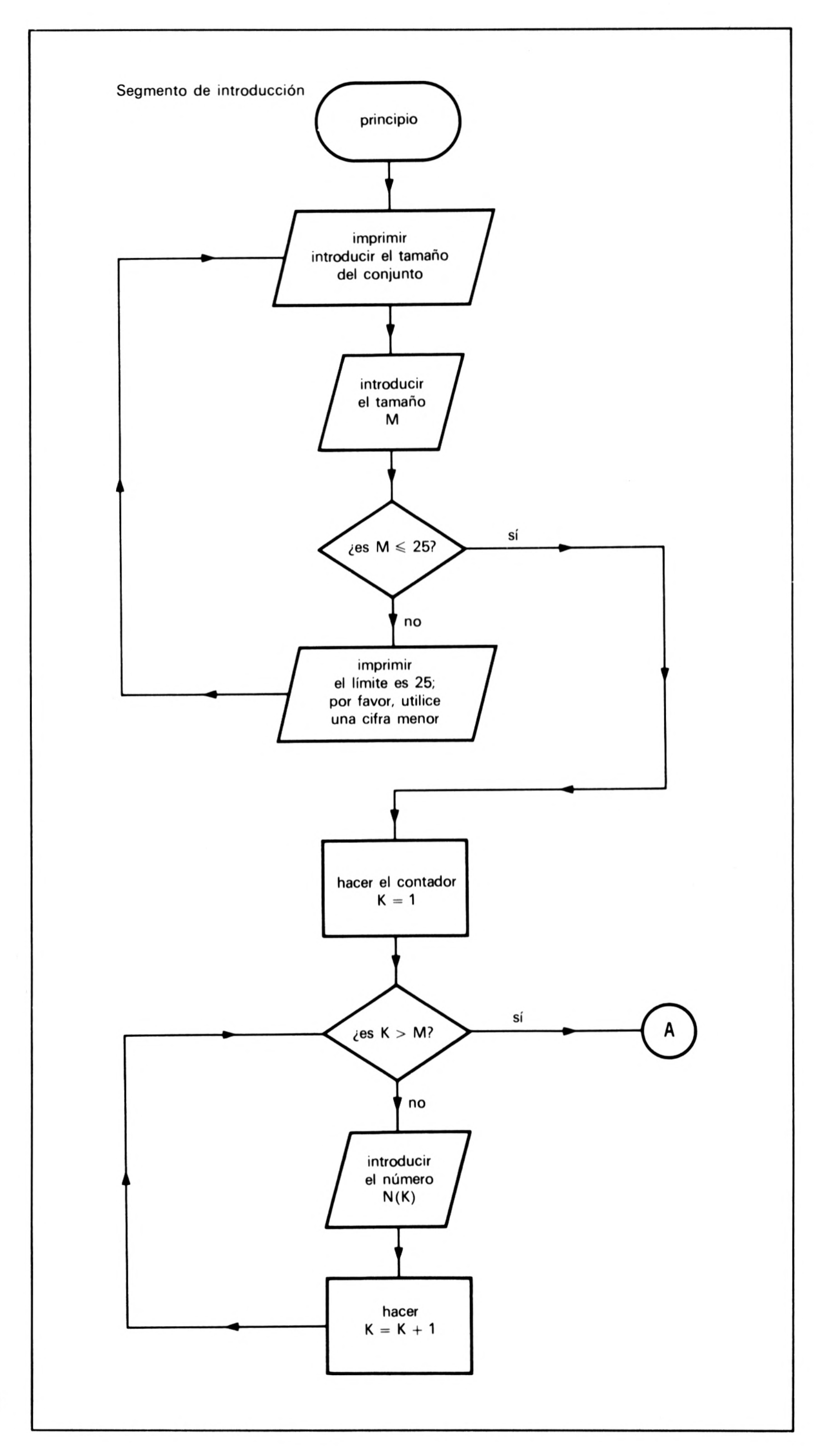
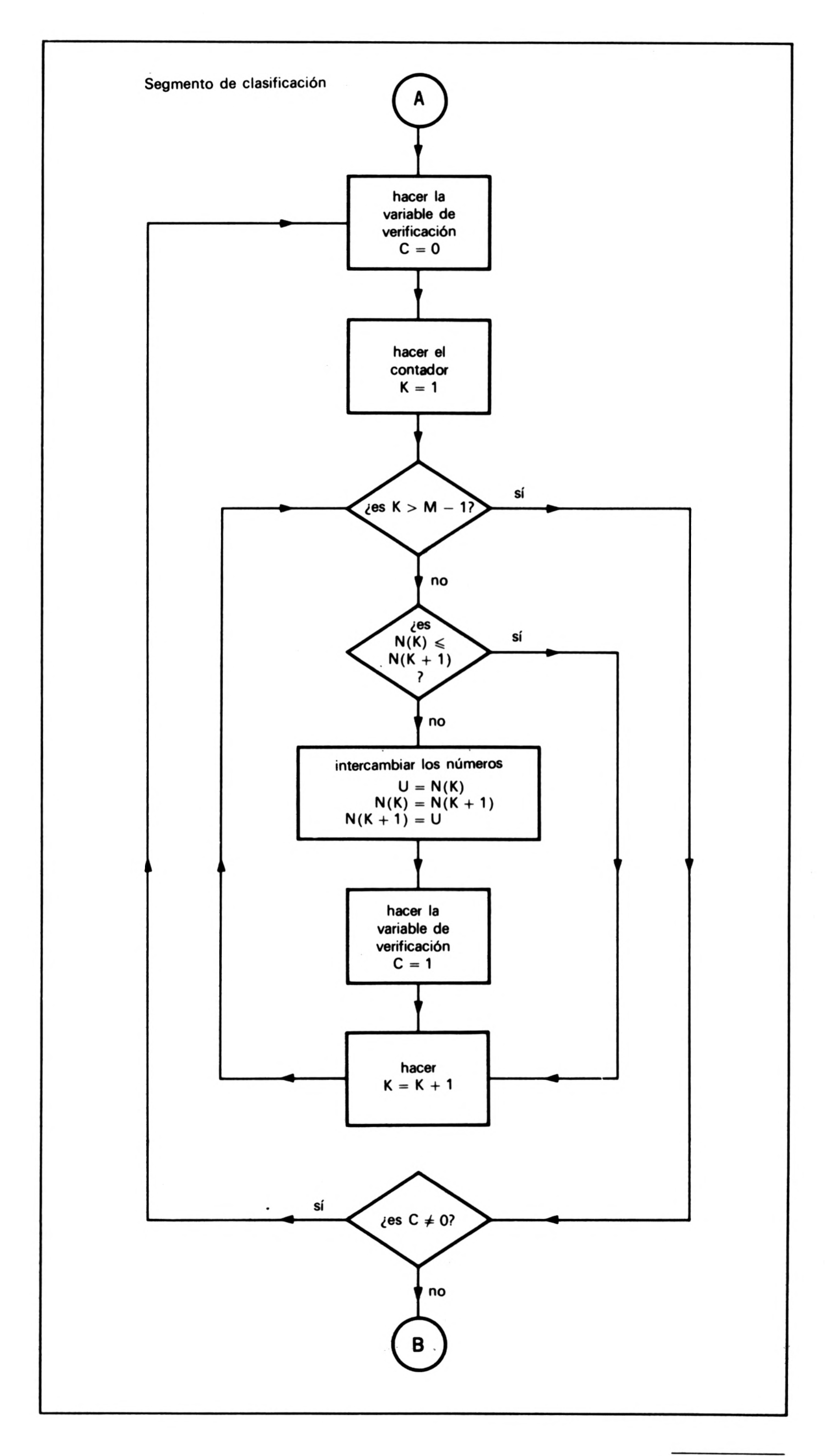
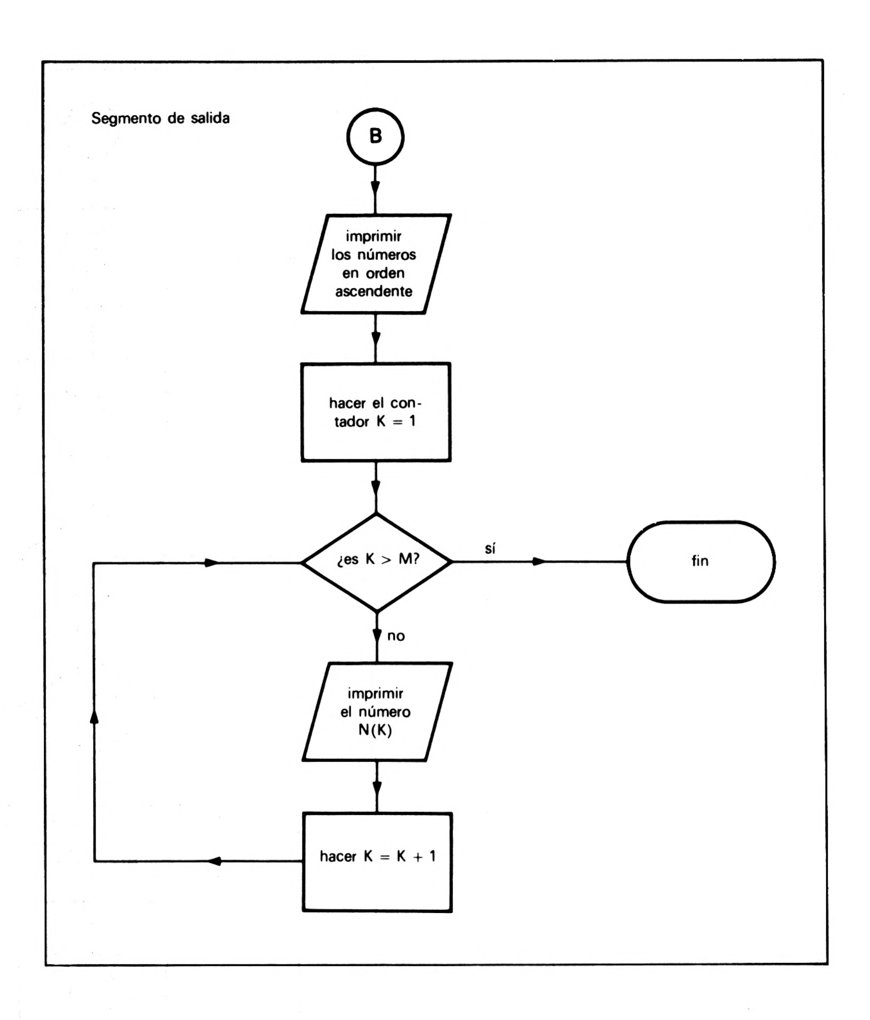


Figura 15.1
Diagrama de flujo del programa ejemplo 15.1



Segmento de clasificación
A
hacer la variable de verificación C = 0
hacer el contador K = 1
¿es K > M − 1?
sí
no
¿es N(K) ≤ N(K + 1) ?
sí
no
intercambiar los números
U = N(K)
N(K) = N(K + 1)
N(K + 1) = U
hacer la variable de verificación C = 1
hacer K = K + 1
sí
¿es C ≠ 0?
no
B




```
500 REM
600 REM SEGMENTO DE SALIDA
610 PRINT "NUMEROS EN ORDEN CRECIENTE"
620 FOR K = 1 TO M
630 PRINT N(K),
640 NEXT K
650 PRINT
660 END
```

Puntos de interés

- La línea 430 debe estudiarse con detenimiento. Tal como está escrita, establece la condición para que los números de un par *no* se intercambien. La condición es que el primer número sea menor *o igual* que el segundo. Si esta condición no se escribe

correctamente, los números iguales se intercambiarían continuamente, y el programa no terminaría nunca.

- Observe la brevedad del segmento de programa, brevedad un tanto engañosa, porque su ejecución puede llevar mucho tiempo.

15.5 Conclusión

En este capítulo se ha expuesto la idea de ordenación y se han discutido algunos problemas prácticos relacionados con ella. También se ha descrito una técnica de ordenación —el método burbuja— que a continuación se ha ejecutado mediante un programa.

Pero el material presentado no agota el capítulo, ni mucho menos. En el ejercicio que sigue se mide la eficacia de la técnica del método burbuja y se modifica el programa propuesto para mejorar su rendimiento. Por último, se introduce otro método de **ordenación por selección.**

La finalidad de estos ejercicios es mejorar la experiencia del lector en los campos de la modificación, la verificación y el diseño de programas, a la vez que mejora su conocimiento de la ordenación.

15 Ejercicio

1. El tiempo que necesita un programa de ordenación por el método burbuja para ordenar una serie de números es proporcional al número de intercambios que debe realizar.

a) Modifique el programa para contar el número de intercambios con arreglo al siguiente método: utilice una variable **J** para el número de intercambios. Haga **J** igual a cero antes de empezar la ordenación utilizando para ello la línea 405. Cada vez que se haga un intercambio, incremente **J** en una unidad mediante la línea 475. Utilice la línea 655 para mostrar el valor final de **J**.

b) Prepare una serie de datos de prueba formados, por ejemplo, por 10 elementos, para comprobar el efecto del grado de orden de los datos introducidos sobre el número de intercambios. Incluya conjuntos de datos al azar, en orden descendente, en orden ascendente e iguales. Pruebe el programa para todos ellos, anote el número de intercambios obtenido en cada caso y comente los resultados.

c) Prepare una serie de datos de prueba para investigar la relación entre el número de elementos del conjunto y el número de intercambios necesario para ordenarlo. Pruebe todos los datos en el programa y anote los intercambios realizados. Haga un gráfico que recoja el número de intercambios frente al número de elementos del conjunto y comente los resultados.

2. La finalidad de este ejercicio es modificar el programa de ordenación por el método burbuja para mejorar su rendimiento.

Estudie las fases del proceso del método burbuja aplicado al siguiente conjunto de datos:

7 4 2 9 3 5

Tras la primera secuencia de comparaciones, el orden será

4 2 7 3 5 9

Observe que el número mayor ocupa el último lugar. Tras la segunda tanda de comparaciones, el orden será

2 4 3 5 7 9

El segundo número en magnitud es ahora el penúltimo.

Cada tanda de comparaciones reduce la parte no ordenada del conjunto en un elemento, ya que al final de aquél se va formando una secuencia ordenada. El algoritmo de ordenación puede modificarse como sigue:

Igualar a cero la longitud de la parte ordenada del conjunto.
Repetir el proceso:
  Comparar cada uno de los pares consecutivos de números de la parte no ordenada del conjunto.
  Si los números de un par no están en orden, intercambiarlos.
  Aumentar la longitud de la parte ordenada del conjunto en 1.
Hasta que todos los pares estén en orden.

a) Modificar la versión del programa de ordenación por el método burbuja escrita en el ejercicio 1 para tener en cuenta esta corrección con arreglo a las siguientes instrucciones:

Represente la longitud de la parte ordenada con la variable **P**.
Ponga **P** a cero en el punto adecuado del programa.
Modifique la línea 420 para tener en cuenta el valor de **P**.
Incremente **P** en 1 en el punto adecuado del programa.

b) Repita los apartados *b*) y *c*) del ejercicio 1 para determinar el efecto de la modificación propuesta.

3. El algoritmo expuesto a continuación describe otro método de ordenación llamado **ordenación por selección:**

Un conjunto de números consta de una **parte ordenada** (al principio de longitud cero) seguida de otra **no ordenada.**
Seleccione el menor número de la parte no ordenada del conjunto.
Colóquelo al principio de esa parte no ordenada.
Retrase un paso el límite de la zona no ordenada (por encima del número que acaba de elegirse).
Hasta que no quede ningún número en la porción no ordenada del conjunto.

a) Diseñe un programa que lleve a la práctica este algoritmo, siguiendo para ello el proceso de refinamiento por etapas.

b) Corrija el programa para que registre el número de veces que se intercambian los números. Utilice los apartados *b*) y *c*) de la pregunta 1 para comparar este programa con el de ordenación por el método burbuja y comente los resultados.

4. Modifique los programas de ordenación por el método burbuja o por selección para ordenar palabras en vez de números. Observe que la relación numérica "menor que" equivale a la alfabética "anterior a".

5. Redacte la documentación para el programador y para el usuario correspondiente a la versión del programa de ordenación por el método burbuja obtenido al final del ejercicio 2.

6. Escriba la documentación para el programador y para el usuario correspondiente al programa de ordenación por selección del ejercicio 4.

7. Un procedimiento muy útil para ordenar datos en un conjunto parcialmente ordenado es el conocido como **barajado;** el resultado de esta ordenación se llama a veces **tabla de barajado.** El proceso tiene dos ventajas: que es bastante rápido y que los datos se localizan en la tabla de picado por el mismo método utilizado para almacenarlos en ella.

Esencialmente, el picado consiste en asociar el valor de un dato a la posición que ocupa en un conjunto parcialmente ordenado. La técnica conoce numerosas variantes, y a continuación se describe una particularmente sencilla.

Para almacenar aproximadamente en orden un conjunto de datos alfabéticos, basta utilizar la letra inicial de cada uno de ellos para determinar su posición en la matriz. Si ésta tiene **M** elementos, la posición de uno en particular viene dada por la fórmula.

```
P = INT(V * M/26)
```

llamada también **algoritmo de barajado;** en ella, **P** es la posición del elemento en la matriz y **V** el valor de la letra inicial (A = 1, B = 2, etc.). Si la posición determinada por esta fórmula ya está ocupada, se utiliza la primera vacía que se encuentre a continuación.

a) Diseñe un programa que acepte la introducción de una serie de datos alfabéticos, los almacene en una tabla de barajado y lleve ésta a la salida. Se sugiere la siguiente estructura general:

> Declarar una matriz y llenar los elementos con un valor adecuado que signifique "vacío".
> Repetir el proceso:
>
> > Introducir un dato.
> > Calcular su posición en la matriz mediante la fórmula anterior.
> > Repetir el proceso:
> >
> > > Comprobar el dato de la posición requerida.
> > > Si la posición está ocupada, probar en la siguiente.
> >
> > Hasta encontrar una posición vacía.
> > Almacenar el dato.
>
> Hasta que la tabla esté llena.
> Mostrar la tabla.

Observe que la búsqueda de una posición vacía debe retroceder desde el final de la matriz hasta el principio de la misma.

b) Modifique el programa para contar el número de comprobaciones necesarias para encontrar un sitio vacío para un dato. Calcule y mues-

tre el número total de comprobaciones para todo el conjunto de datos.

Prepare varios datos de prueba, todos de la misma longitud, para investigar el efecto del orden de los datos de entrada sobre el número de verificaciones necesarias para encontrar lugares de almacenamiento vacíos y sobre el grado de orden de la tabla de barajado resultante.

8. Diseñe un algoritmo de barajado para clasificar datos numéricos. Incorpórelo a una versión modificada del programa de creación de una tabla de barajado.
9. Redacte un programa que lea la agenda telefónica personal del programa ejemplo 10.1, clasifique los registros por orden alfabético de nombres y transfiera los registros clasificados a otro fichero.


RESET
RETURN

16
Búsqueda

En este capítulo estudiaremos el problema de la localización de un dato dentro de un conjunto amplio de ellos, un proceso conocido como **búsqueda.** Se discutirán algunas ideas generales y algunas técnicas de localización, y se diseñará un programa para llevar a la práctica la técnica de búsqueda más frecuente. Este mismo programa se verifica en uno de los ejercicios propuestos al final del capítulo, ejercicios que darán también la oportunidad de diseñar programas para ejecutar otras técnicas de búsqueda.

También en este capítulo haremos uso de los métodos de diseño, verificación, modificación y documentación de programas expuestos en otros anteriores. La búsqueda es una técnica de programación fundamental con gran cantidad de aplicaciones.

16.1
Concepto de búsqueda

La idea general de búsqueda es sencilla: consiste en localizar un dato o un grupo de datos dentro de un conjunto de ellos más amplio. Los datos pueden ser aislados, pero lo más frecuente es localizar un registro dentro de un fichero.

Aunque la búsqueda no plantea los problemas de la ordenación, puede también resultar difícil. La situación más complicada se da cuando el conjunto de datos en que hay que buscar se encuentra en una memoria auxiliar y es demasiado amplio como para caber en la memoria central del ordenador.

Aquí hemos evitado deliberadamente esta clase de problemas, y se estudia la búsqueda en relación con grupos de datos sencillos y reducidos. En el capítulo 22 se mencionan algunas técnicas de búsqueda en grandes conjuntos de datos. De todas formas, las técnicas expuestas aquí pueden ampliarse para manipular los mencionados grupos amplios.

16.2
Técnicas de búsqueda

Hay dos técnicas de uso común: la **búsqueda secuencial** y la **búsqueda binaria.** La primera sirve para cualquier conjunto de datos, mientras que la segunda sólo es compatible con datos ordenados.

Si se dispone de un conjunto de datos parcialmente ordenado por la técnica de barajado, la búsqueda puede efectuarse siguiendo un procedimiento muy similar al empleado para ordenarlos. Este método se discutirá en uno de los ejercicios propuestos al final del capítulo. Las próximas secciones van dedicadas al estudio de las técnicas secuencial y binaria.

16.3
Búsqueda secuencial

Esta operación consiste en examinar un conjunto de datos uno por uno hasta dar con el deseado. Tiene varias ventajas y algunos inconvenientes graves.

Entre las ventajas cabe citar que se trata de un proceso muy sencillo, aplicable por igual a conjuntos de datos ordenados y sin ordenar. Si los datos se encuentran en una memoria auxiliar que sólo permite el acceso secuencial, dato a dato o registro a registro, es evidente que la técnica de elección será la búsqueda secuencial.

El inconveniente principal es la lentitud del proceso, ya que, por término medio, es preciso recorrer más de la mitad de los datos almacenados antes de encontrar el que se busca.

En Basic, el acceso a los ficheros es secuencial, registro a registro y, por tanto, la búsqueda secuencial es el proceso obvio de localización de un registro dentro de un fichero. El programa ejemplo 10.2 realiza

precisamente la búsqueda secuencial de un fichero, por lo que en este capítulo no estudiaremos ningún otro programa del mismo tipo.

16.4 Búsqueda binaria

La **búsqueda binaria,** también llamada **dicotómica,** consiste en la reducción sistemática del conjunto de datos en el que se encuentra el elemento buscado. La técnica sólo es aplicable a conjuntos de datos ordenados.

Descrito de manera informal, el método es el siguiente:

En principio, el dato buscado puede estar en cualquier parte del conjunto general.
Se examina el dato central; si coincide con el deseado, el proceso termina.
En caso contrario, y si el dato central es mayor que el deseado, éste se encuentra en la primera mitad del conjunto.
Si el dato central es menor que el deseado, éste se encuentra en la segunda mitad del conjunto.
En cualquiera de los dos últimos casos, se repite el proceso sobre la mitad del conjunto seleccionada.

Como la porción del conjunto en que se encuentra el dato deseado se reduce a la mitad en cada paso, la localización es bastante rápida incluso dentro de conjuntos muy amplios.

Condición imprescindible para poder recurrir a esta técnica es que los datos del conjunto, o los registros del fichero, sean accesibles directamente, lo que supone que se encuentren en la memoria central o en una memoria auxiliar de acceso directo, como un disco magnético. El acceso a cada dato, o a cada registro, se produce por medio de una dirección o un índice.

En la sección siguiente estudiaremos un programa que lleva a cabo la búsqueda binaria en un caso muy sencillo: un conjunto ordenado de datos numéricos almacenados en una matriz.

16.5 Programa ejemplo 16.1

Este programa lleva a la práctica la técnica de búsqueda binaria expuesta en la sección anterior. Se aplica a un conjunto ordenado de datos numéricos.

Método

En primer lugar, hay que elaborar un algoritmo que describa la búsqueda binaria de manera más formal que en la sección anterior. Supongamos que el conjunto ordenado de datos numéricos está almacenado en una matriz **N** de longitud **M**. El dato buscado tiene el valor **X**. El área en que se encuentra el dato deseado está limitada por los índices **L** y **U**.

El algoritmo es el siguiente:

Igualar **L** a **1** y **U** a **M**.

Repetir el proceso:

> Localizar el dato central, de índice
> **I = INT((L + U)/2)**.
> Si **X > N(I)** hacer **L = I + 1** (**X** está en la primera mitad).
> Si **X < N(I)** hacer **U = I − 1** (**X** está en la segunda mitad).

Hasta que **X = N(I)**, porque entonces se habrá localizado el dato;
o hasta que **L** sea mayor que **U**, porque en tal caso el dato no se encuentra en el conjunto.

Observe que hay dos condiciones de final del bucle, que corresponden, respectivamente, a la culminación fructífera e infructuosa del proceso de búsqueda.

Diseño del programa

El programa supone que se dispone ya de un conjunto ordenado de datos almacenado en una matriz **N** de longitud **M**. Por tanto, no incluye el módulo de introducción o creación de tal estructura de datos. Así, la descomposición inicial del programa en pasos será:

Introducir un dato para su localización.
Llevar a cabo el proceso de búsqueda binaria.
Mostrar el resultado.

El único refinamiento posible es introducir el algoritmo de búsqueda de la sección anterior y ampliar la sección de salida:

Introducir un dato para su localización.
Fijar los valores iniciales del área que ha de investigarse.
Repetir el proceso de búsqueda sobre áreas cada vez menores.

Si se encuentra el dato deseado,
mostrar su posición;
en caso contrario, comunicar mediante un mensaje que la búsqueda ha sido infructuosa.

La tarea a realizar está ya suficientemente detallada como para redactar el programa.

Variables

N(25)	Matriz de números ordenados en orden ascendente.
M	Cantidad de números de la matriz.
L	Límite inferior del área de búsqueda.
U	Límite superior del área de búsqueda.
I	Punto central del área de búsqueda.
X	Número que ha de localizarse.

Diagrama de flujo

La figura 16.1 ilustra el flujo de control del programa.

Programa

```
1000 REM PROGRAMA EJEMPLO 16.1
1010 REM BUSQUEDA BINARIA
1020 DIM N(25)
1100 REM MODULO DE INTRODUCCION
1110 REM UNA SERIE ORDENADA DE DATOS DE LONGITUD M
1120 REM EN UNA MATRIZ N
1130 REM
1200 REM SEGMENTO DE BUSQUEDA
1210 PRINT "NUMERO QUE HA DE LOCALIZARSE?";
1220 INPUT X
1230 LET L = 1
1240 LET U = M
1250 LET I = INT ((L + U)/2)
1260 IF X = N(I) THEN 1430
1270 IF X > N(I) THEN 1300
1280 LET U = I - 1
1290 GOTO 1310
1300 LET L = I + 1
1310 IF L <= U THEN 1250
1400 REM SEGMENTO DE SALIDA
1410 PRINT "EL NUMERO NO ESTA EN EL CONJUNTO"
1420 STOP
1430 PRINT "EL NUMERO SE ENCUENTRA EN LA POSICION
           "; I
1440 STOP
1450 END
```

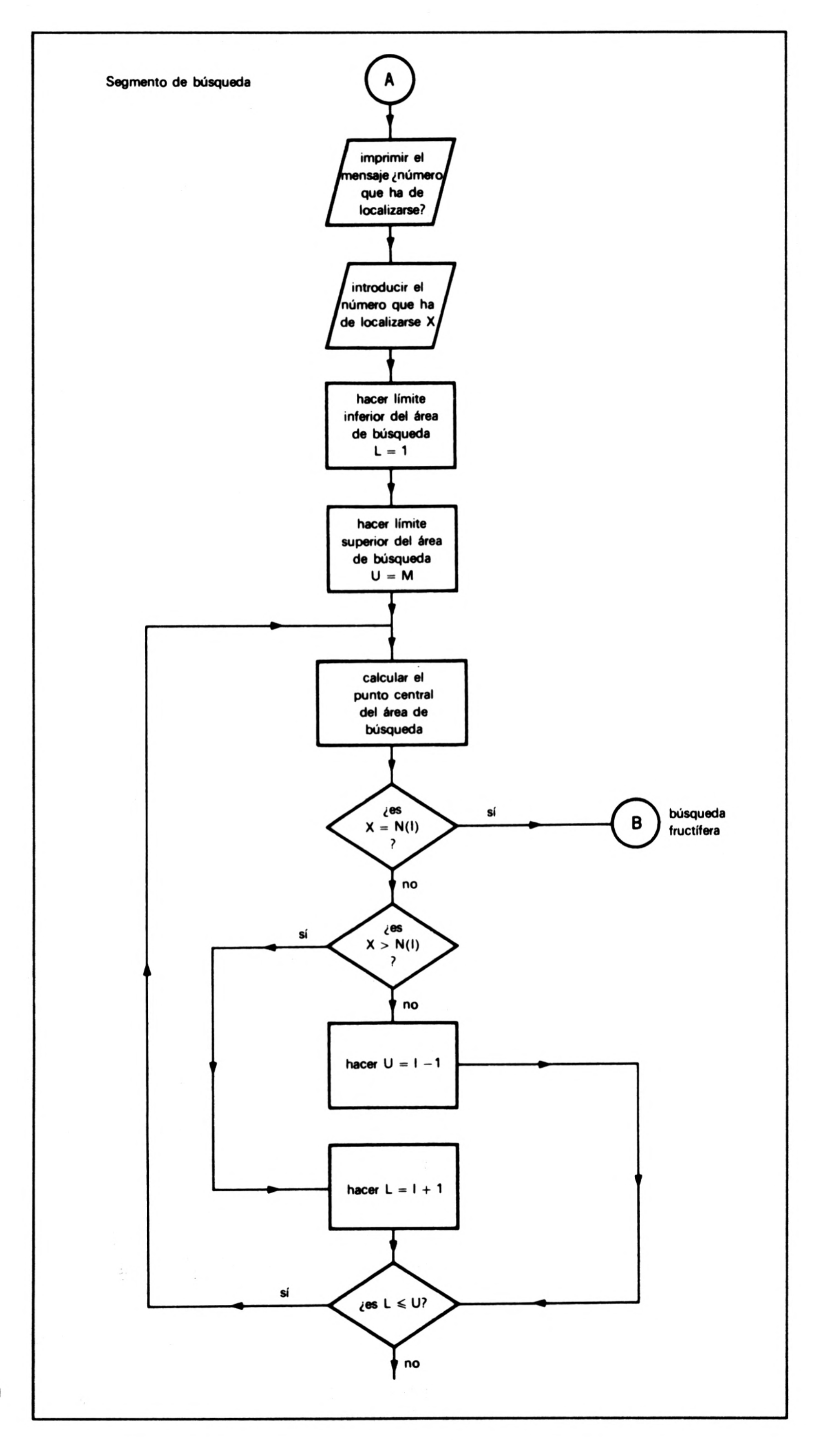
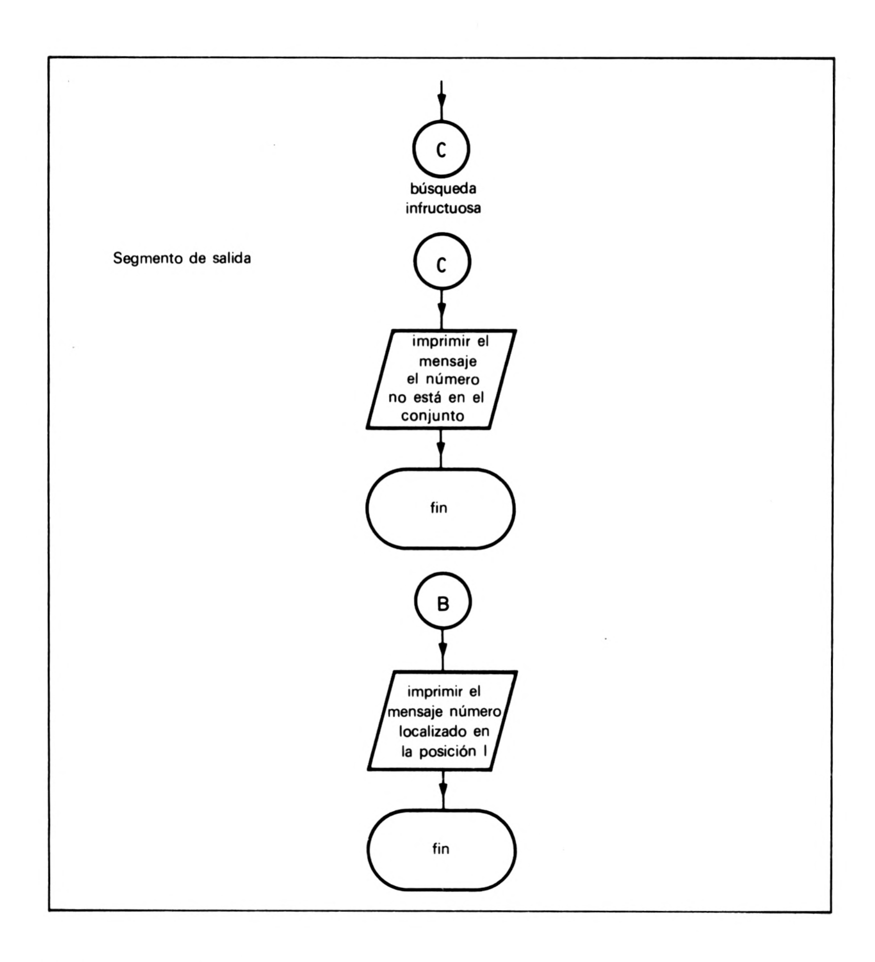


Figura 16.1
Diagrama de flujo del programa ejemplo 16.1



Puntos de interés

- El programa es una interpretación bastante libre del algoritmo "repetir hasta que", porque una de las condiciones de fin del bucle se verifica al principio del mismo, en la línea 1260.
- Aunque tal como está escrito es un programa bastante conciso, puede abreviarse todavía más utilizando una versión ampliada del Basic.

16.6 Conclusión

Hemos presentado en este capítulo la idea de búsqueda y un par de técnicas —secuencial y binaria— para llevarla a cabo.

En los ejercicios que siguen se medirá la eficacia del programa de

búsqueda binaria y se comparará con la técnica de búsqueda secuencial. También se estudia la localización de un dato dentro de un conjunto ordenado mediante la técnica de barajado.

16 Ejercicio

1. Diseñe y escriba un programa que realice la búsqueda secuencial dentro de un conjunto ordenado de datos almacenados en una matriz **N** que contiene **M** elementos.

2. Escriba un módulo de entrada sencillo para usar con el programa ejemplo 16.1 y con el del ejercicio 1 anterior. El módulo ha de permitir introducir el valor de la variable **M**, comprobar si excede o no de los límites de la matriz e introducir a continuación en ésta **M** valores. Los valores numéricos se introducirán en orden ascendente.

3. El tiempo que emplea un programa de búsqueda en completar su tarea depende del número de datos que deba comparar con el deseado hasta localizar el equivalente a éste.

a) Modifique el programa ejemplo 16.1 y el escrito como respuesta al ejercicio 1, de forma que cuenten el número de datos comparados con el que se busca. El resultado debe aparecer en la salida junto con el dato localizado.

b) Diseñe un conjunto de datos (de 10 elementos, por ejemplo) apropiado para verificar los dos programas. Localice uno tras otro todos los elementos del conjunto y anote el número de comparaciones que han sido necesarias en cada caso. Trate también de localizar elementos ausentes del conjunto. Calcule el número medio de comparaciones para cada uno de los programas.

c) Diseñe varios conjuntos de datos que contengan, por ejemplo, 4, 8, 16 y 32 elementos y utilícelos para verificar el funcionamiento de los dos programas. Trace un gráfico que ilustre el número de comparaciones frente al número de elementos del conjunto.

4. Modifique el programa ejemplo 16.1 de manera que funcione con datos alfanuméricos en lugar de numéricos.

5. Redacte la documentación para el programador y la guía para el usuario correspondientes al programa ejemplo 16.1.

6. *a*) Escriba un programa para localizar un dato contenido en una tabla de barajado (ejercicio 7 del capítulo 15).

Introduzca los elementos de la tabla y el dato deseado. Utilice el mismo algoritmo empleado para crear la tabla, para localizar la posición inicial en que podría encontrarse el dato y para recorrer a continuación varias posiciones sucesivas hasta encontrarlo.

b) Modifique el programa de manera que cuente el número de comprobaciones necesarias para localizar un dato. Utilizando algunas de las tablas de barajado producidas por el programa correspondiente, ejecute varias veces el programa. Observe el efecto del grado de ordenación de la tabla de barajado sobre el número de comprobaciones necesarias para localizar un dato.

7. Vuelva a escribir el programa ejemplo 16.1 en una versión ampliada del Basic utilizando la construcción **IF ... THEN ... ELSE** e incluyendo varias instrucciones en una sola línea.

8. Escriba un programa capaz de buscar un nombre en la agenda telefónica ordenada (ejercicio 9 del capítulo 15) reproduciendo el fichero completo en una o más matrices de la memoria principal y siguiendo una técnica de búsqueda binaria.


RESET
RETURN

17
Refundición*

Este es el último de los capítulos dedicados a operaciones de programación fundamentales. Presenta el concepto de **refundición** y una técnica para llevarlo a cabo en Basic. Como en los capítulos previos de esta parte, se hace uso de las técnicas de diseño, comprobación y documentación de programas.

17.1 Concepto de refundición

Refundir es combinar dos conjuntos ordenados de datos para producir uno solo, también ordenado. En la práctica, los datos a refundir son casi siempre registros ordenados por su **clave** y pertenecientes a uno o más ficheros. Pero en este capítulo examinaremos una situación sencilla: la refundición de conjuntos de datos individuales almacenados en matrices.

Si dos conjuntos de datos están formados por los siguientes elementos:

Conjunto A	*Conjunto B*
34621	29415
79114	30217
83047	69815
98216	80427
	96118

* MERGE en inglés.

el conjunto refundido quedará así:

29415
30217
34621
69815
79114
80427
83047
96118
98216

Observe que no es necesario que los dos conjuntos de partida tengan idéntica longitud, aunque sí deben estar en el mismo orden (ascendente o descendente).

La refundición es un proceso muy útil por sí mismo, aunque una de sus aplicaciones más importantes es la ordenación de conjuntos de datos demasiado grandes para la memoria central del ordenador. Esta aplicación se discutirá en la sección 17.4.

17.2 Una técnica de refundición

En comparación con la ordenación y la búsqueda, la refundición es una operación relativamente sencilla. A continuación se expone un algoritmo de refundición del tipo más general y en la sección siguiente se hará una versión del mismo más específica para obtener a partir de ella un programa de refundición en Basic.

Sean dos cadenas alfanuméricas de datos ordenados, llamadas **Cadena A** y **Cadena B**, que desean refundirse en una cadena única. Ambas cadenas disponen de un indicador de fin de datos. El algoritmo general de refundición sería:

Leer un dato de cada una de las cadenas de entrada.
Mientras no se llegue al indicador de fin de datos en las dos cadenas.
Repetir el proceso:

Si se ha alcanzado el indicador de fin de datos de la Cadena A, **Copiar de B.**
Si se ha alcanzado el indicador de fin de datos de la Cadena B, **Copiar de A.**
Si el elemento en curso de la Cadena A es menor que el correspondiente de la Cadena B,
Copiar de A.
En caso contrario, **Copiar de B.**

Escribir el indicador de fin de datos en la cadena de salida.

Siendo

Copiar de A: pasar el dato en curso de la Cadena A a la cadena refundida. Leer el siguiente dato de la Cadena A.
Copiar de B: pasar el dato en curso de la Cadena B a la cadena refundida. Leer el siguiente dato de la Cadena A.

Como se deduce del algoritmo, no es preciso ocupar con grandes cantidades de datos la memoria central del ordenador, porque tanto las cadenas de entrada como la de salida pueden archivarse en una memoria auxiliar. Los ficheros se leen y escriben secuencialmente.

17.3 Programa ejemplo 17.1

El objetivo de este programa es refundir dos conjuntos ordenados de datos numéricos almacenados en matrices para dar lugar a un tercer conjunto de datos, almacenado también en una matriz. Todos los conjuntos terminan con un indicador de fin de datos.

Esta propuesta constituye un desarrollo práctico de la idea de refundición extraordinariamente simple, pero que de todas formas sirve para revelar lo esencial de la técnica. En las próximas secciones y en los ejercicios propuestos al final del capítulo se considerarán algunas ampliaciones del programa.

Aunque el algoritmo sugiere el empleo de la construcción **IF ... THEN ... ELSE**, no la usaremos aquí. El programa está redactado en la versión de referencia del Basic definida al comienzo del libro. Quien lo desee, puede volver a escribirlo en una versión ampliada.

Diseño del programa

El programa supone que las cadenas alfanuméricas de entrada están ya cargadas en otras tantas matrices, por lo que la actividad de "máximo nivel" desarrollada por el mismo es:

Refundir los datos de las matrices de entrada en una matriz de salida.

Para descomponer esta tarea en fases se utiliza el algoritmo de la sección anterior, pero con algunas modificaciones para tener en

cuenta el hecho de que los datos están almacenados en matrices. Estas se leen o se escriben por medio de tres indicadores, que representan en cada una el índice del dato en curso. Antes de nada daremos nombres a las variables, para a continuación volver a escribir el algoritmo definitivo, ya con los nombres correspondientes.

Variables

A, B, C Matrices para las cadenas de entrada y la cadena de salida.
A1, B1 Dato en curso en las matrices de entrada.
I, J, K Indicadores de las matrices de entrada y salida.

Los indicadores de fin de datos tienen el valor 99999.

Algoritmo corregido

Corregido para adaptarlo a las variables y estructuras de datos elegidas, el algoritmo de refundición quedaría como sigue:

Llevar los indicadores **I**, **J** y **K** a 1.
Hacer **A1** = **A(I)** y **B1** = **B(J)**.
Mientras no se cumpla **A1** = 99999 y **B1** = 99999.
Repetir el proceso:

Si **A1** = 99999, **copiar de B.**
En caso contrario, si **B1** = 99999, **copiar de A.**
En caso contrario, si **A1** < **B1**, **copiar de A.**
En caso contrario, **copiar de B.**

Almacenar 99999 en la matriz **C**.

Siendo

Copiar de A: hacer **C(K)** = **A1**, incrementar **I** y **K** en 1, hacer **A1** = **A(I)**.
Copiar de B: hacer **C(K)** = **B1**, incrementar **J** y **K** en 1, hacer **B1** = **B(J)**.

Este algoritmo está ya suficientemente detallado como para poder escribir el programa a partir de él.

Diagrama de flujo

La figura 17.1 ilustra el flujo de control del programa ejemplo 17.1.

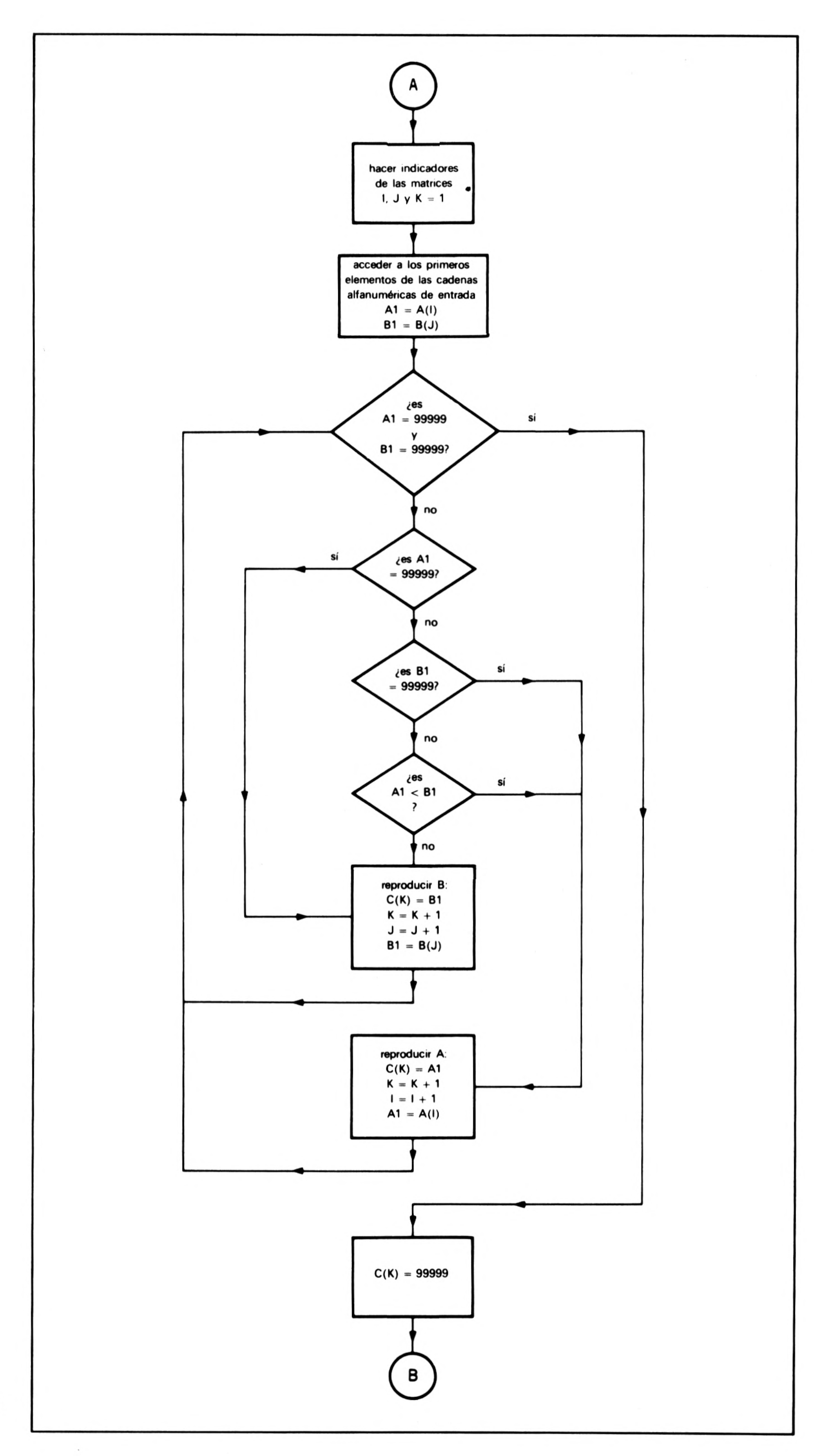


Figura 17.1
Diagrama de flujo del programa ejemplo 17.1

Programa

```
1000 REM PROGRAMA EJEMPLO 17.1
1005 REM REFUNDICION DE DOS SERIES DE DATOS
1010 REM LOS MODULOS DE ENTRADA Y SALIDA NO ESTAN
         INCLUIDOS
1015 DIM A(25), B(25), C(50)
1020 REM
1100 LET I = 1
1105 LET J = 1
1110 LET K = 1
1115 LET A1 = A(I)
1120 LET B1 = B(J)
1125 IF A1 = 99999 AND B1 = 99999 THEN 1205
1130 IF A1 = 99999 THEN 1145
1135 IF B1 = 99999 THEN 1175
1140 IF A1 < B1 THEN 1175
1145 REM COPIAR DE B
1150 LET C(K) = B1
1155 LET K = K + 1
1160 LET J = J + 1
1165 LET B1 = B(J)
1170 GOTO 1125
1175 REM COPIAR DE A
1180 LET C(K) = A1
1185 LET K = K + 1
1190 LET I = I + 1
1195 LET A1 = A(I)
1200 GOTO 1125
1205 LET C(K) = 99999
1210 END
```

Puntos de interés

- Gran parte del programa se dedica a verificar si se ha llegado o no al final de los datos de cada una de las cadenas alfanuméricas.
- Este programa se condensaría notablemente haciendo uso de la construcción **IF ... THEN ... ELSE** y de la posibilidad de reunir varias instrucciones en una sola línea.

17.4 Refundición y ordenación

Como ya se dijo al principio de este capítulo, la refundición puede utilizarse para ordenar conjuntos de datos excesivamente amplios para la capacidad de la memoria principal del ordenador. Hay varias técnicas de **refundición-ordenación,** pero todas se basan en los siguientes principios generales:

- El conjunto de datos, en la mayor parte de los casos un fichero, se divide en cadenas suficientemente pequeñas para que quepan en la memoria central del ordenador.
- Las cadenas se extraen de la memoria auxiliar una a una y se ordenan en la central. Las cadenas ordenadas vuelven a reproducirse en la memoria auxiliar.
- Los pares de cadenas ordenadas se refunden para producir cadenas más largas, y el proceso se repite hasta que todos los datos se hayan refundido en una cadena única.

El principio de la refundición-ordenación puede aplicarse también a los datos contenidos en la memoria principal. Los datos se consideran como un gran número de cadenas de longitud unitaria. Estas se refunden en pares para producir cadenas de longitud doble, que a su vez vuelven a refundirse hasta que todos los datos quedan juntos en una sola cadena. Se trata de una técnica de ordenación bastante compleja, y además el número de elementos a ordenar debe ser una potencia entera de dos (2, 4, 8, 16, 32, etc.).

Tales técnicas son un tanto elevadas para un curso de este nivel, pero de todas formas en el ejercicio propuesto al final del capítulo se plantean algunas cuestiones dirigidas a los que estén interesados en llevarlas a la práctica.

17.5 Conclusión

En este capítulo se ha expuesto la idea de refundición y se ha discutido su aplicación a la ordenación de grandes conjuntos de datos. También se ha diseñado un programa de refundición.

Aquí acaba la parte del texto dedicada a operaciones fundamentales de programación. Las técnicas de ordenación, búsqueda y refundición se usan muchísimo y se llevarán a la práctica en algunos de los capítulos finales, dedicados a aplicaciones de la programación.

17 Ejercicio

1. Defina brevemente los siguientes términos: refundición, refundición-ordenación, clave, cadena.

2. Escriba el contenido de la matriz **C** producida por el programa ejemplo 17.1 en cada uno de los siguientes casos:

a) La matriz **A** tiene los elementos: 40000, 60000, 99999.
La matriz **B** tiene los elementos: 30000, 50000, 99999.

b) Las matrices **A** y **B** tienen únicamente el valor 99999.
c) La matriz **A** tiene los elementos: 30000, 20000, 99999.
La matriz **B** tiene los elementos: 10000, 50000, 99999.
d) La matriz **A** tiene los elementos: 10000, 10001.
La matriz **B** tiene los elementos: 99999, 99999.

3. Escriba las condiciones bajo las que el programa ejemplo 17.1 producirá resultados correctos y terminará adecuadamente.

4. Vuelva a escribir el programa ejemplo 17.1 haciendo uso de la construcción **IF ... THEN ... ELSE** y agrupando varias instrucciones en una sola línea.

5. Si dispone de un ordenador capaz de leer simultáneamente varios ficheros de datos, escriba un programa para refundir datos de dos ficheros de entrada en uno solo de salida. Utilice el algoritmo de la sección 17.2 como base del programa.

Utilice ficheros con un solo dato en cada registro o con varios, situación ésta más realista. En este caso, uno de los datos constituirá la clave del registro, que determina el orden en los ficheros de entrada y, en consecuencia, el orden de refundición.

6. Si dispone de un ordenador capaz de leer simultáneamente varios ficheros de datos, escriba un programa sencillo de refundición-ordenación con arreglo a las siguientes indicaciones:

1) Elija una cadena alfanumérica de longitud suficiente para representar el máximo número de datos almacenables en la memoria del ordenador.
2) Cree cuatro ficheros en memoria auxiliar, que llamaremos aquí **A**, **B**, **C** y **D**.
3) Escriba en el fichero **A** un conjunto de datos no ordenados de cuatro cadenas de longitud.
4) Reproduzca los datos, cadena a cadena, en una matriz de la memoria central y ordénelos. Reproduzca las cadenas ordenadas en los ficheros **C**, **D**, **C** y **D** (en este orden). Los ficheros **C** y **D** contienen cada uno dos cadenas ordenadas.
5) Refundir la primera cadena del fichero **C** con la primera del fichero **D**, y transferir una cadena ordenada de longitud doble al fichero **B**.
6) Refundir la segunda cadena del fichero **C** con la segunda del **D** y transferir una cadena ordenada de longitud doble al fichero **A**.
7) Refundir las cadenas ordenadas de longitud doble de los ficheros **A** y **B** para formar una cadena ordenada única en el **C**.

Sugerencias

1) Marque todas las cadenas con un indicador de fin de datos.
2) Utilice los pasos de arriba como módulo de control en el nivel superior de la estructura del programa.
3) Modifique el programa de refundición escrito para la pregunta 5, para convertirlo en un subprograma llamado por el módulo de control. Los parámetros de transferencia incluyen los nombres de los ficheros de origen y destino.
4) Modifique el programa ejemplo 15.1 para convertirlo en un subprograma de ordenación de una cadena de datos de una matriz.
5) Escriba subprogramas para introducir datos, cargarlos en un fichero, leerlos de un fichero para llevarlos a una matriz y extraerlos de una matriz para escribirlos en un fichero. Los parámetros incluyen en todos los casos el nombre del fichero.

7. Escriba un programa de refundición-ordenación para un conjunto de datos almacenados en una matriz. El número de elementos de ésta debe ser una potencia entera de 2 (32 y 64 son los más adecuados en este caso).

Hacen falta también dos matrices de trabajo, dos de longitud igual a la mitad de la original y una tercera de esa misma longitud.

Base el programa en los siguientes principios generales:

1) El programa realiza varios **pasos de refundición,** en número igual a la potencia de 2 correspondiente al número de elementos (5 para 32 elementos, 6 para 64).
2) Durante cada paso de refundición, se unen varias cadenas alfanuméricas de la matriz para constituir otras de longitud doble. En el primer paso se refunden cadenas de un elemento para formar otras de 2; en el segundo paso, se refunden éstas en otras de 4 elementos, etc.
3) Cada refundición individual recorre las siguientes etapas:

   Copiar de las dos cadenas que van a refundirse en las dos matrices de trabajo más cortas.

   Refundir las dos cadenas en la matriz de trabajo más larga.

   Transferir la cadena refundida a la posición ocupada por las dos cadenas originales en la matriz principal.

8. Compare el programa diseñado y escrito como respuesta a la pregunta 7 con el programa ejemplo 15.1 de ordenación por el método burbuja y comente los siguientes aspectos:

a) Velocidades relativas de trabajo de dichos programas.
b) Efecto del grado de orden de los datos de entrada sobre los tiempos de ejecución de cada uno de los programas.
c) Necesidades de memoria de cada uno de los programas.


RESET
RETURN

18
Pilas

En informática es muy frecuente manipular los datos no como elementos individuales, sino como estructuras. En los capítulos anteriores hemos examinado ya varias de tales estructuras: matrices, ficheros y registros.

En esta parte del texto analizaremos cuatro estructuras muy importantes: pilas, colas, listas y árboles. Todas ellas están caracterizadas por propiedades definitorias sencillas. A cada una se dedicará un capítulo, que describirá sus propiedades y enseñará a llevarlas a la práctica en un programa escrito en Basic. Este primero se centrará sobre la más elemental de tales estructuras: la **pila.**

Para trabajar con estas estructuras de datos hay que emplear una variable peculiar llamada **puntero,** que denota la posición de otra variable en la memoria del ordenador. Como los punteros no pueden definirse en Basic, todas las estructuras de datos estudiadas en estos capítulos se crearán dentro de matrices. El índice de un elemento revela su posición dentro de la matriz; lo más parecido a un puntero que puede conseguirse en Basic es una variable que represente un valor de ese índice.

18.1

Propiedades de las pilas

Una **pila,** conocida también como **lista directa,** es una colección de datos que únicamente pueden extraerse o introducirse por un extremo. Por ello se dice que una pila es una estructura del tipo **último-en-entrar—primero-en-salir (UEPSA,** LIFO en inglés). El extremo por el que se accede a la pila se llama **cima.**

Una pila sólo admite dos operaciones: añadir un nuevo elemento por la cima, acción conocida como **introducir** o **apilar,** y extraer o **vaciar** un elemento, también por la cima; en este caso, el elemento situado a continuación del extraído pasa a ocupar la cima.

Cuando se almacena una pila en la memoria del ordenador, sus elementos ocupan posiciones consecutivas y el puntero señala la cima. Dicho puntero se "desplaza" cada vez que se extrae o se introduce un elemento. El otro extremo de la pila, llamado **base,** está fijo.

18.2

Representación de una pila en Basic

La única forma posible de representar una pila en Basic es almacenar sus elementos en una matriz, lo que impone una primera limitación: la pila no puede nunca hacerse mayor que la matriz.

Es práctica común en informática situar la base de la pila en el último elemento de la matriz, el identificado por un índice más alto. De esta forma, la pila va creciendo "hacia arriba" en la matriz, y está llena cuando llega al primer elemento.

El puntero de la pila equivale al índice del primer elemento libre de la matriz, situado sobre la cima de la pila, como suele ser habitual en proceso de datos; de esa forma es fácil identificar una pila vacía. La realización práctica del concepto de pila expuesto en esta sección se ilustra en la figura 18.1.

18.3

Programa ejemplo 18.1

El objetivo de este programa es crear una pila y ejecutar las operaciones de introducción y extracción de elementos.

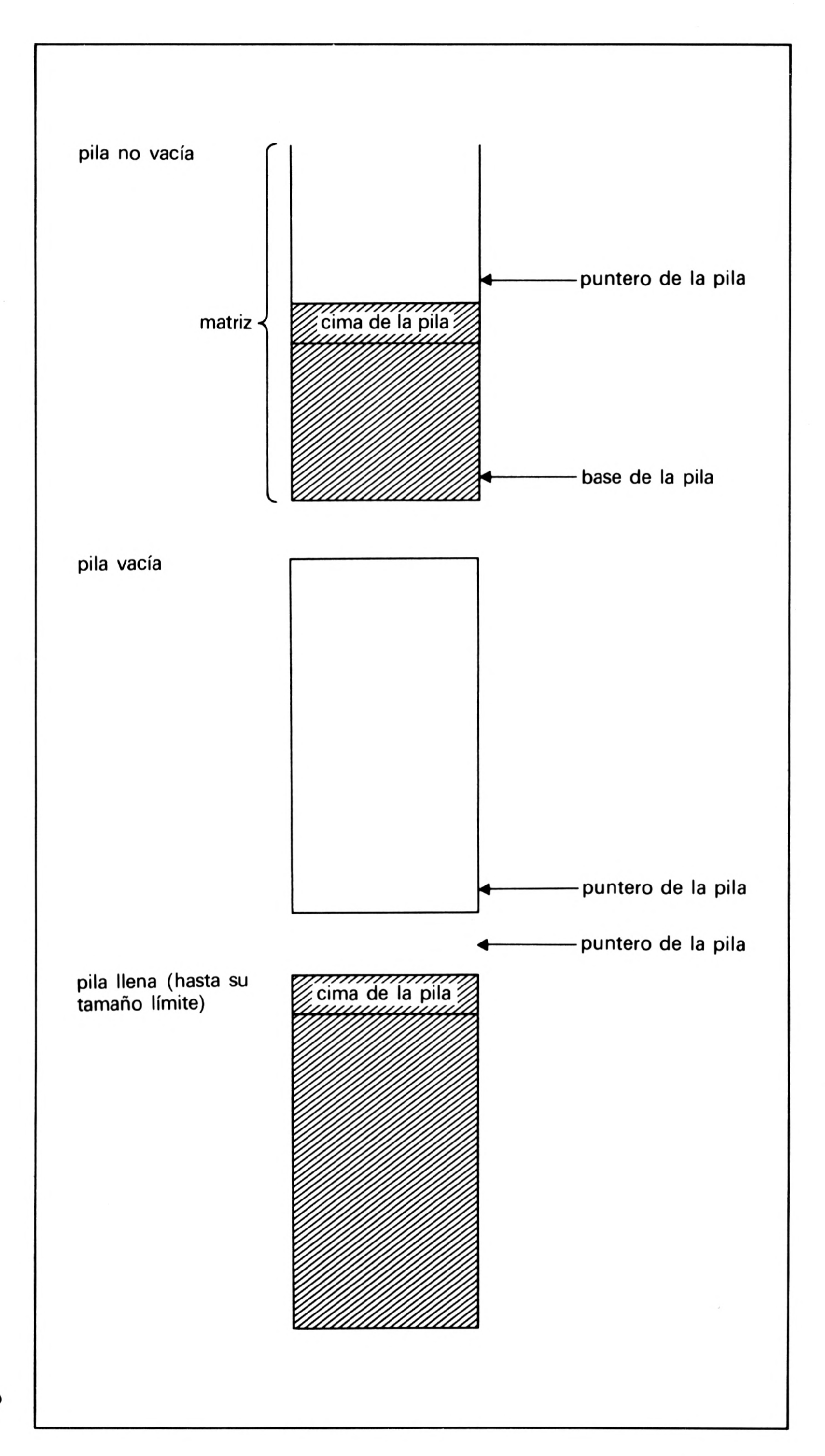


Figura 18.1
Plasmación práctica del concepto de pila

Diseño del programa

El programa está organizado en tres módulos, que funcionan como subprogramas controlados por un programa principal que hace uso de la pila. Dicho programa principal no se incluye aquí. Los módulos realizan las siguientes tareas:

- crear una pila vacía,
- introducir un elemento dado en la pila,
- vaciar la pila y entregar el valor del elemento de la cima.

Como ya se ha dicho en la sección anterior, la pila se almacena en una matriz. Dado que en Basic una matriz no puede transferirse como parámetro a un subprograma, los tres módulos actúan sobre una matriz común y comparten un único puntero de pila. Los parámetros intercambiados entre los dos últimos módulos son datos introducidos o extraídos más una variable que indica el buen o mal resultado de la operación.

La segunda etapa del diseño del programa consiste en especificar la acción de cada uno de los módulos con cierto detalle.

Creación de una pila vacía

Este módulo lleva el puntero de la pila al máximo valor del índice de la matriz. Según lo expuesto en la sección 18.2, esto indica una pila vacía.

Introducir un elemento dado en la pila

El funcionamiento general del módulo es el siguiente:

Si la pila está llena,

señalar el fallo y volver;
en caso contrario, insertar el elemento en la posición señalada
por el puntero de la pila;
reducir el valor del puntero en 1,
confirmar el éxito y volver.

Según lo expuesto en la sección anterior, la pila está llena cuando el puntero adopta un valor inferior a 1.

Extracción del elemento superior de la pila

El funcionamiento general del módulo es el siguiente:

Si la pila está vacía,

señalar el fallo y volver;
en caso contrario, incrementar en 1 el valor del puntero;
reproducir el elemento de la posición señalada por el puntero de la pila en el parámetro de retorno;
confirmar el éxito y volver.

La pila está vacía si el puntero adopta el valor máximo del índice de la matriz. Ahora los módulos ya están descritos con detalle suficiente como para escribir el programa.

Variables

Globales:

S(25) Matriz de 25 elementos para almacenar la pila.
P Puntero de la pila.

Parámetros:

U Variable introducida en la pila.
D Variable extraída de la pila.
F Indicador de resultado: **F** = 1, buen resultado; **F** = 0, malo.

Diagrama de flujo

La figura 18.2 ilustra el flujo de control de los tres módulos del programa ejemplo 18.1.

Módulos del programa

```
1000 REM PROGRAMA EJEMPLO 18.1
1005 REM MANIPULACION DE LA PILA
1010 REM VARIABLES GLOBALES
1015 REM S: MATRIZ PARA ALMACENAR LA PILA
1020 REM P: PUNTERO DE LA PILA
1025 DIM S(25)
1030 REM
```

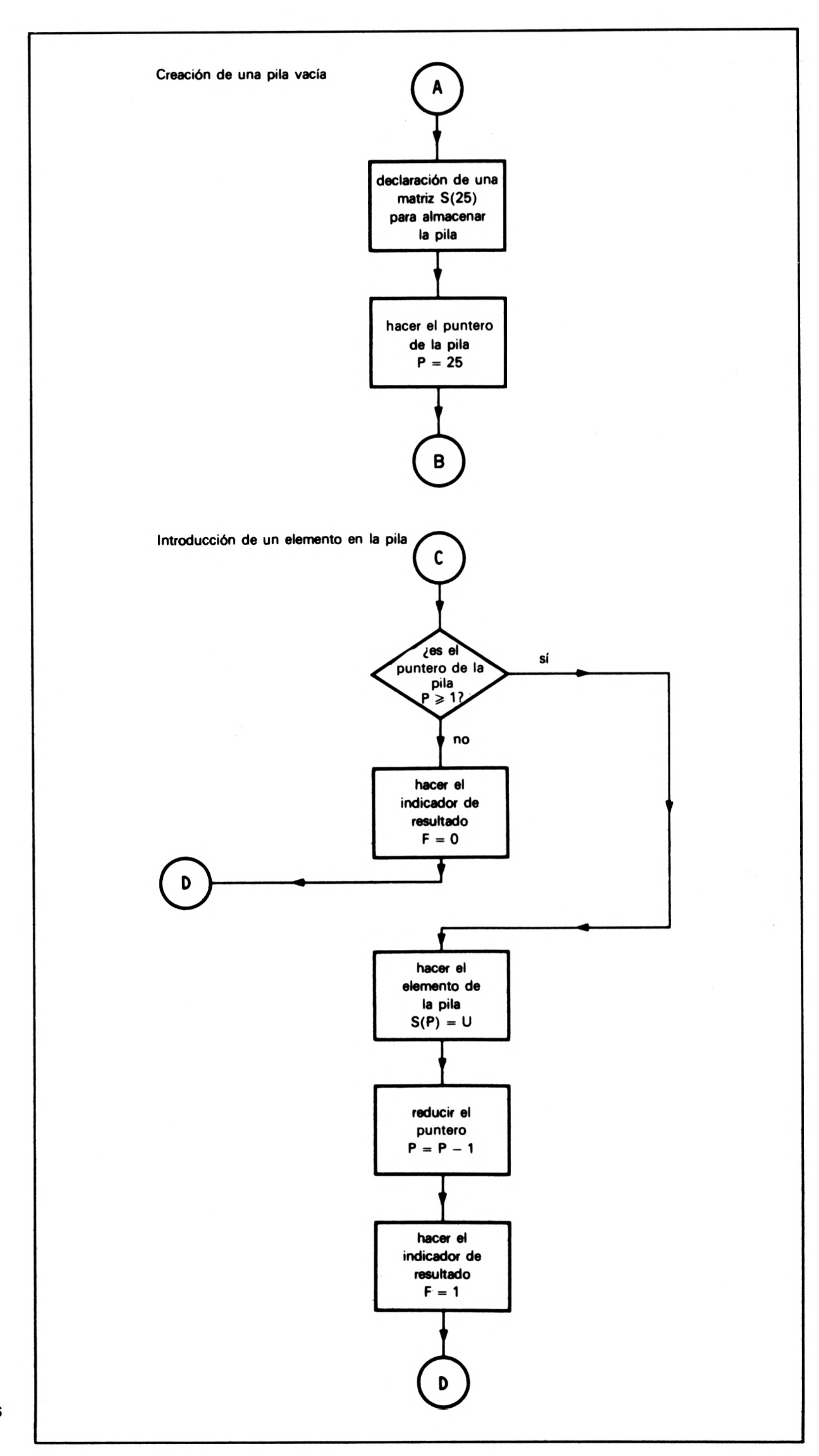
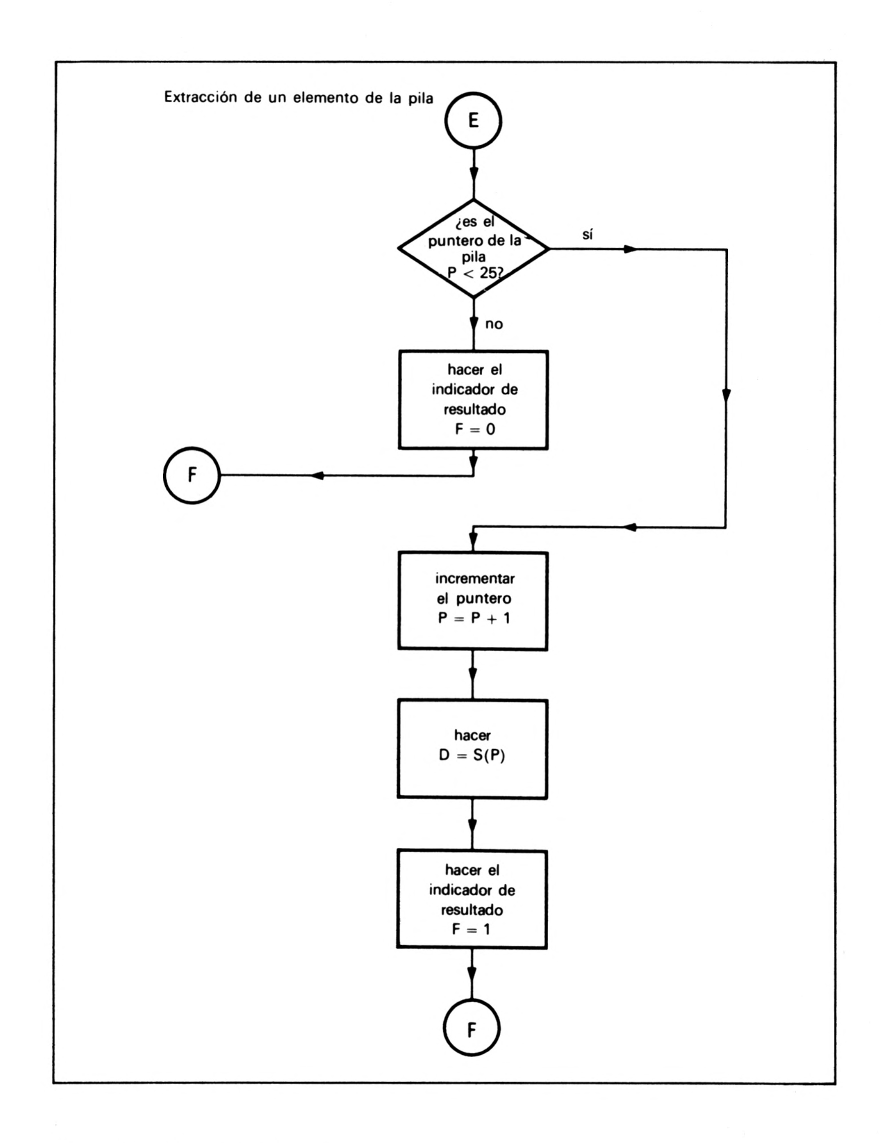


Figura 18.2
Diagrama de flujo de los módulos del programa ejemplo 18.1



Creación de una pila vacía

```
2000 REM CREACION DE UNA PILA VACIA
2005 REM SIN PARAMETROS
2010 LET P = 25
2015 RETURN
```

Introducción de un elemento en la pila

```
2100 REM INTRODUCCION DE UN ELEMENTO EN LA PILA
2105 REM PARAMETROS
2110 REM U: ELEMENTO INTRODUCIDO EN LA PILA
```

```
2115 REM F: INDICADOR DE ACIERTO/FALLO
2120 IF P >= 1 THEN 2135
2125 LET F = 0
2130 RETURN
2135 LET S(P) = U
2140 LET P = P - 1
2145 LET F = 1
2150 RETURN
2155 REM
```

Extracción de un elemento de la pila

```
2200 REM EXTRACCION DE UN ELEMENTO DE LA PILA
2205 REM PARAMETROS
2210 REM D: ELEMENTO
2215 REM F: INDICADOR DE ACIERTO/FALLO
2220 IF P <= 25 THEN 2235
2225 LET F = 0
2230 RETURN
2235 LET P = P + 1
2240 LET D = S(P)
2245 LET F = 1
2250 RETURN
2255 REM
```

Puntos de interés

- En los módulos de introducción y extracción se ha invertido la condición establecida por el algoritmo para simplificar la estructura del programa.
- Si cualquiera de las dos operaciones —introducción o extracción— no puede ejecutarse, la pila permanece invariable.

18.4 Algunas aplicaciones de las pilas

Las pilas se usan mucho en informática. Aparecen en ocasiones en programas de aplicaciones, pero se emplean sobre todo en programas del sistema: compiladores, sistemas operativos y programas de utilidad. A estos niveles, las pilas se emplean para llevar registros detallados del avance de los cálculos, para seguir ciertos aspectos de la traducción de un lenguaje de programación a otro, para transferir información de un programa principal a los subprogramas y para resolver interrupciones. En muchos ordenadores actuales, la memoria

central está organizada como una pila controlada por uno o más registros que actúan de punteros. En los ejercicios del final del capítulo se investigan más de cerca algunas de estas aplicaciones.

18.5 Conclusión

En este capítulo hemos estudiado una estructura de datos tan sencilla como potente: la pila. Se han discutido sus propiedades generales y se han escrito varios módulos de un programa para llevarlas a la práctica. Los puntos más importantes son:

- Un puntero es una variable que denota la posición de otra variable en la memoria del ordenador.
- Una pila es una colección de datos que sólo pueden introducirse o extraerse por un extremo. Es una estructura del tipo último-en-entrar—primero-en-salir (UEPSA).
- El extremo accesible de la pila es la cima de la misma y el fijo constituye su base.
- En cualquier realización práctica de una pila, basta un solo puntero para señalar la posición real de la cima.
- La mayor parte de las aplicaciones de la pila se encuentran a nivel del soporte lógico del sistema.

En el ejercicio que se propone a continuación se investigarán algunas aplicaciones de las pilas.

18 Ejercicio

1. Defina brevemente los siguientes términos: puntero, pila, cima de la pila, base de la pila, puntero de la pila, introducir, extraer, UEPSA.
2. Indique las variaciones en el contenido de una pila inicialmente vacía tras cada una de las operaciones de la siguiente secuencia:

   introducir 5
   introducir 7
   extraer
   introducir 9
   introducir 11
   extraer
   extraer
   extraer

3. Escriba otro módulo para el programa de manipulación de pilas que indique el valor en curso de la cima de la pila, pero sin extraerlo. Si la pila está vacía, se produce un mensaje de fallo.
4. Es posible realizar secuencias de cálculos utilizando una pila para almacenar los resultados intermedios. Para dirigir una secuencia de operaciones realizada de esa forma se ha creado la llamada **notación polaca inversa.**

He aquí algunos ejemplos de dicha notación:

Notación normal	*Notación polaca inversa*
$3 + 5$	$3\ 5\ +$
$2 \times (7 + 4)$	$2\ 7\ 4\ +\ \times$
$(3 - 2) \times 4 + (6 + 9) \div 8$	$3\ 2\ -\ 4\ \times\ 6\ 9\ +\ 8\ \div\ +$

Observe que la notación polaca inversa hace innecesarios los paréntesis.

La interpretación de una secuencia de cálculos representados en notación polaca inversa se realiza como sigue:

Seguir la secuencia de izquierda a derecha.
Si se encuentra un número, introducirlo en la pila.
Si se encuentra un signo de operación,

extraer los dos números superiores de la pila,
realizar la operación con esos dos números,
introducir la respuesta en la pila.

El único número que habrá en la pila al terminar la secuencia de operaciones será el resultado.

He aquí un ejemplo resuelto:

Cálculo: $2\ 7\ 4\ +\ \times$

Pasos	*Pila*
introducir 2	2
introducir 7	7 2
introducir 4	4 7 2
+	11 2
×	22

El resultado es 22.

Diseñe y escriba un programa para introducir una cadena de caracteres en notación polaca inversa que represente un cálculo y realizar dicho cálculo. Lleve el resultado a la salida. Para simplificar el programa, es aconsejable trabajar con números de una sola cifra.

Estructure el programa en tres niveles: el nivel superior, ocupado por el **módulo de control**, se encarga de la introducción de las cadenas de caracteres alfanuméricos y las examina carácter por carácter. A continuación muestra el resultado del cálculo.

Cada vez que se detecta un número o un signo de operación, el control pasa a un **módulo de operación**, situado en un segundo nivel. Si algún carácter resulta imposible de identificar, se presenta un mensaje y se interrumpe el cálculo.

En el segundo nivel hay un módulo por cada operación aritmética y otro más encargado de identificar cada uno de los números de la cadena de caracteres, para introducirlo en la pila. Estos módulos utilizan el subprograma de introducción y extracción del programa ejemplo 18.1.

En el tercer nivel se sitúan los subprogramas del programa ejemplo 18.1 y el programa escrito como respuesta a la pregunta 3. Este último módulo se utiliza para transferir el resultado del cálculo al módulo de control. El programa puede modificarse para que funcione repetidamente o para que muestre el valor en curso del extremo superior de la pila tras cada una de las operaciones.

5. Redacte la documentación para el programador y para el usuario, correspondiente al programa escrito en respuesta a la pregunta 4.

6. Vuelva a escribir los módulos del programa ejemplo 18.1 utilizando la construcción **IF ... THEN ... ELSE** e incluyendo varias instrucciones en una sola línea.


RESET
RETURN

19
Colas

Este es el segundo de los capítulos dedicados al estudio de estructuras de datos. En él nos ocuparemos de una muy común en la vida cotidiana: la **cola.** Se presentarán las propiedades que definen esta estructura y a continuación se examinará un programa que las plasma en Basic. Al final se mencionan algunas aplicaciones de las colas. Varias preguntas del ejercicio propuesto al término del capítulo se relacionan con tales aplicaciones.

Se utilizan aquí técnicas ya estudiadas en otras partes del libro, como el diseño, la verificación y la documentación de programas. Por otra parte, las ideas presentadas en este capítulo se aplicarán en algunos de los situados más adelante.

19.1
Propiedades de las colas

Una cola se caracteriza por las siguientes propiedades:

Los elementos de la cola están ordenados. Las aportaciones nuevas se añaden por la parte posterior de la misma, y las

extracciones se efectúan por la anterior. No se pueden introducir elementos por este extremo anterior. Por ello, se dice que una cola es una estructura del tipo **primero-en-entrar—primero-en-salir (PEPSA,** FIFO en inglés).

Como en la pila, en la cola sólo pueden efectuarse dos operaciones: **insertar** un elemento colocándolo en el extremo final y **extraer** un elemento tomándolo del extremo delantero.

Hay varias formas de poner una cola en la memoria del ordenador. Una de ellas consiste en asociar a cada elemento de la misma un indicador que denota la posición del siguiente. Otro procedimiento consiste en alojar los elementos en posiciones consecutivas de la memoria, con punteros situados al principio y al final de la cola; este último método es el que emplearemos en este capítulo, aunque con algunas modificaciones. El anterior se utilizará en el próximo capítulo, dedicado a las listas.

19.2

Representación de una cola en Basic

Como en el caso de la pila, la única forma posible de representar una cola en Basic es almacenar sus elementos en una matriz. Esto impone un límite al tamaño de la cola y plantea un problema adicional que no existía en el caso de la pila: a medida que se van añadiendo elementos por la parte posterior y retirando otros de la anterior, la cola retrocede dentro de la matriz y deja espacios vacíos delante de sí.

La solución al problema es hacer que la cola "dé la vuelta", de manera que la parte posterior de la misma quede arriba dentro de la matriz. Un espacio de memoria que se comporta de esta forma se llama a veces **memoria circular.**

A diferencia de la pila, se acepta que la cola crece "hacia abajo" dentro de la memoria que ocupa, una matriz en este caso, y aquí seguiremos esta convención.

Se utilizan dos punteros. Uno señala la posición del elemento situado en el extremo anterior de la cola y el otro denota el espacio disponible situado en la parte posterior de la misma. De esta forma es fácil saber cuándo una cola está vacía, aunque la matriz no puede llenarse por completo, porque en esa situación no habría forma de diferenciar una cola vacía de otra llena. Por tanto, se considera que la cola está llena cuando en la matriz sólo queda un elemento vacío. La figura 19.1 ilustra algunas colas implementadas de acuerdo con la descripción dada aquí.

19.3

Programa ejemplo 19.1

El objetivo de este programa es crear una cola y realizar las operaciones de inserción y extracción de elementos de la misma.

Diseño del programa

El programa sigue el diseño general del ejemplo 18.1 de manejo de pilas. Hay tres módulos, escritos como subprogramas, que realizan las tres operaciones deseadas, y no se incluye el programa principal. La cola se almacena en la matriz que, al igual que los punteros anterior y posterior, es común a todos los subprogramas. Son parámetros de éstos los valores que han de insertarse o extraerse y el indicador de buen o mal resultado. Especificaremos a continuación los tres módulos con cierto detalle.

Creación de una cola vacía

Este módulo fija los indicadores anterior y posterior en el primer elemento de la matriz. Como ilustra la figura 19.1, esta situación denota una cola vacía.

Inserción en la cola de un elemento dado

El funcionamiento general de este módulo es el siguiente:

Si la cola está llena:

    indicar mal resultado y volver;
    en caso contrario, insertar el elemento en la posición señalada
        por el puntero final;
    si el valor del puntero final es igual al límite de la matriz,
        poner el puntero final a 1;
        en caso contrario, sumar 1 al puntero final;
    indicar buen resultado y volver.

De acuerdo con lo expuesto en la sección anterior, la cola está llena si el puntero final vale 1 menos que el inicial, o si éste vale 1 y el final es igual al límite de la matriz.

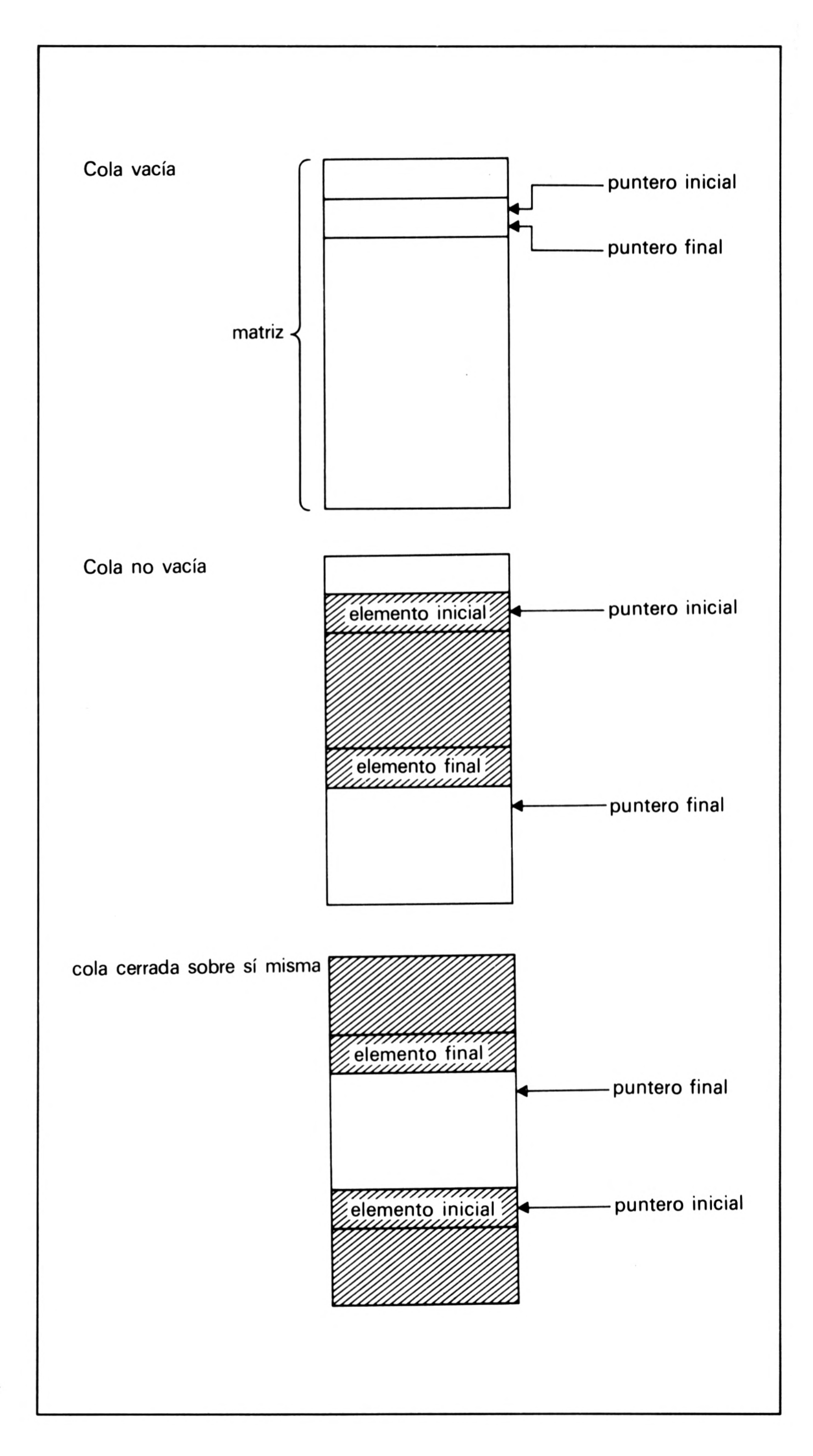
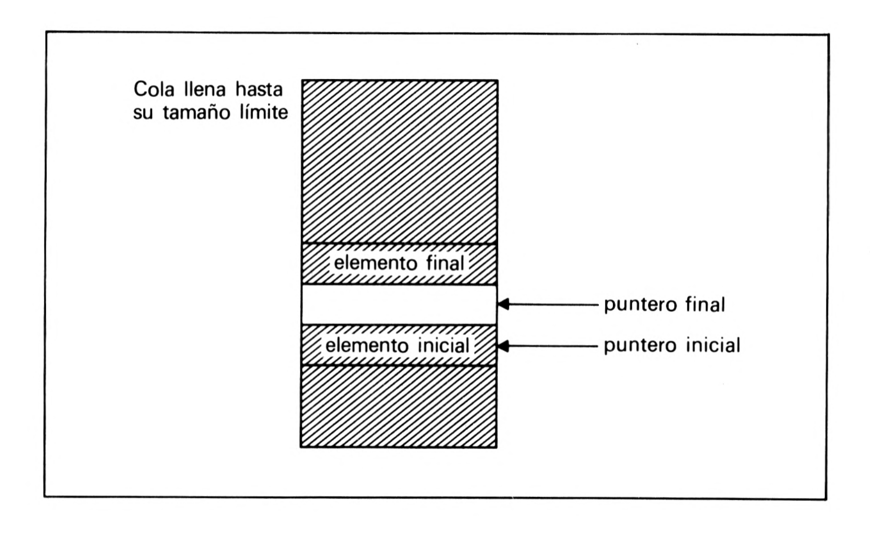


Figura 19.1
Una forma de representar el concepto de cola



Extracción de un elemento de la parte inicial de la cola

El funcionamiento general de este módulo es el siguiente:

Si la cola está vacía:

indicar mal resultado y volver;
en caso contrario, copiar el elemento situado en la posición señalada por el puntero inicial en el parámetro de vuelta;
si el valor del puntero inicial es igual al límite de la matriz,

poner el puntero inicial a 1;
en caso contrario, sumar 1 al puntero inicial;

indicar buen resultado y volver.

De acuerdo con lo expuesto en la sección anterior, la cola está vacía si los punteros inicial y final son iguales. Los módulos ya están descritos con detalle suficiente como para escribir el programa a partir de ellos.

Variables

Globales:

Q(25) Matriz de 25 elementos para almacenar la cola.
I Puntero inicial.
F Puntero final.

Parámetros:

A Elemento insertado en la cola.
B Elemento extraído de la cola.
S Indicador de resultado: S = 1, buen resultado, S = 0, mal resultado.

Diagrama de flujo

La figura 19.2 ilustra el flujo de control de los tres módulos del programa ejemplo 19.1.

Módulos del programa

```
1000 REM PROGRAMA EJEMPLO 19.1
1005 REM MANIPULACION DE UNA COLA
1010 REM VARIABLES GLOBALES
1015 REM Q: MATRIZ PARA ALMACENAR UNA COLA
1020 REM I: PUNTERO INICIAL
1025 REM F: PUNTERO FINAL
1030 DIM Q(25)
1035 REM
```

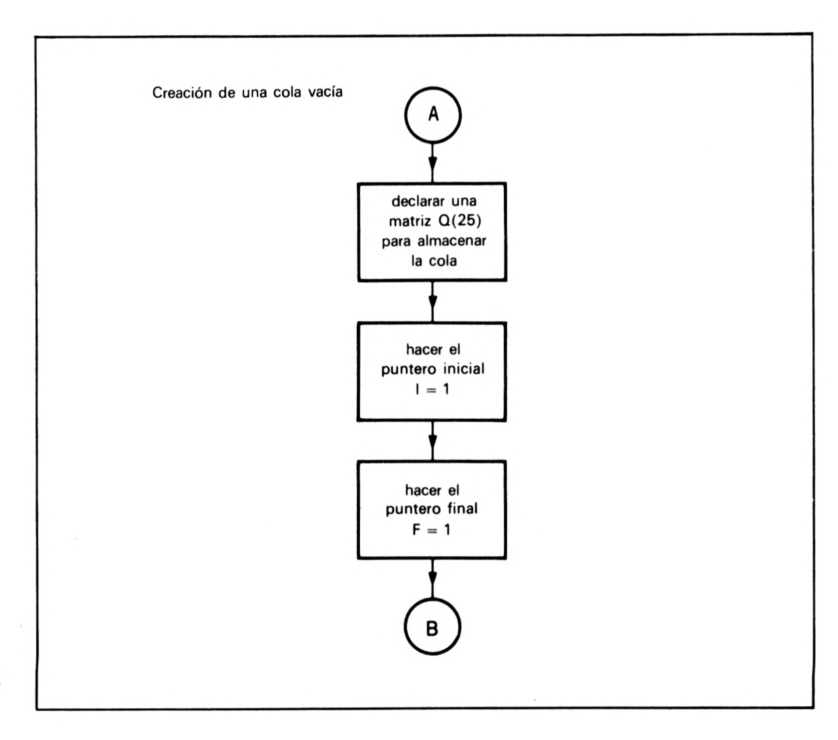
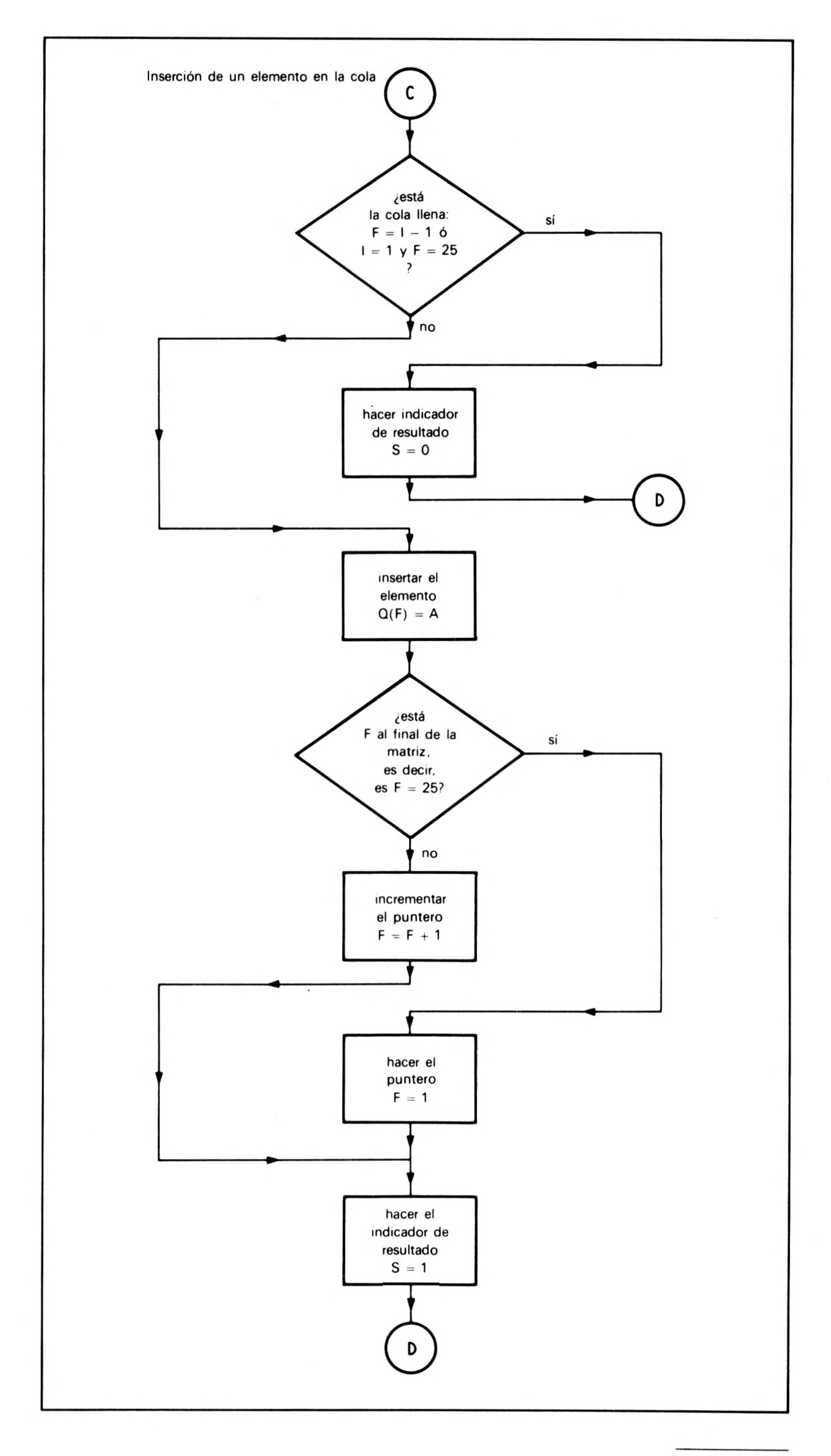
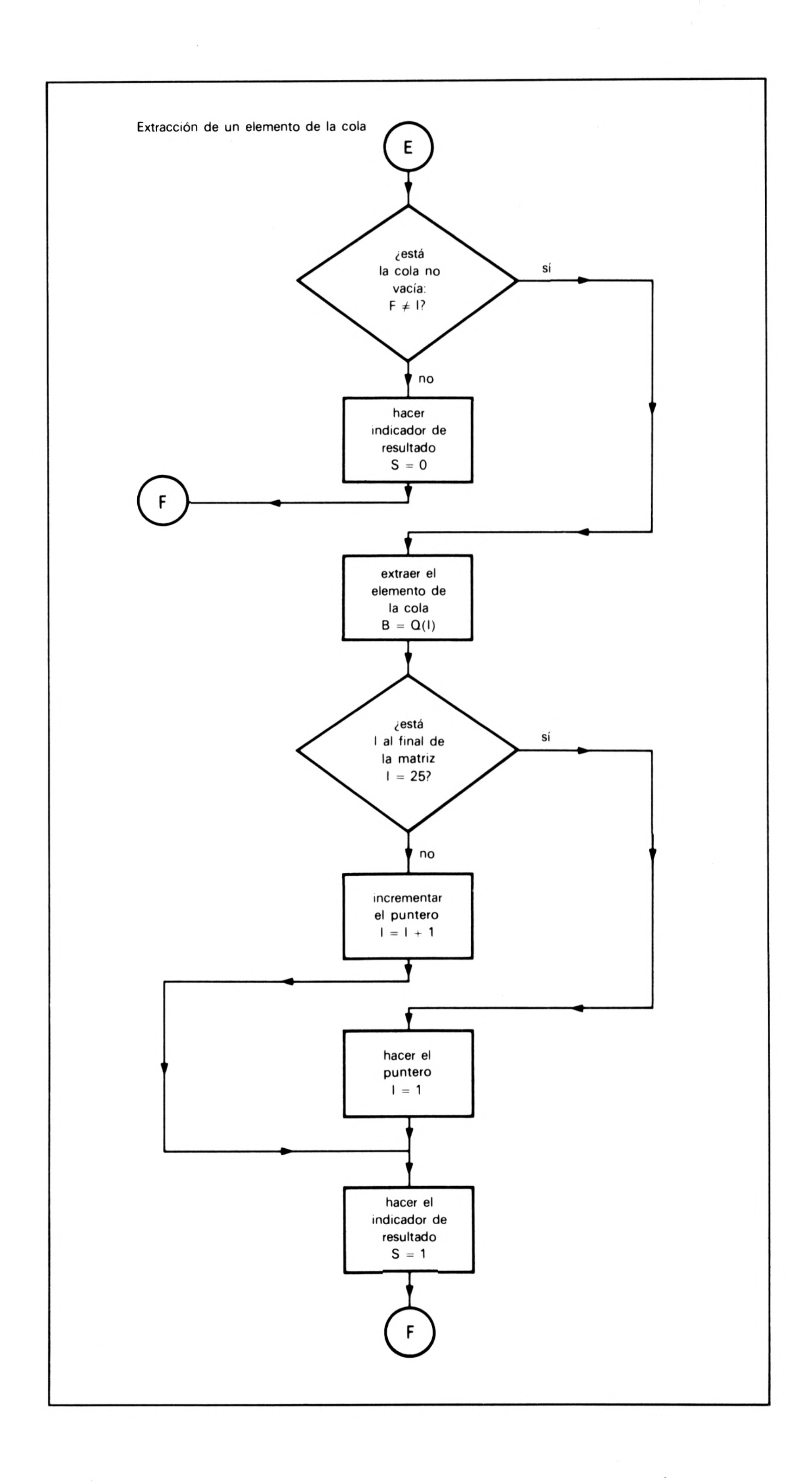


Figura 19.2
Diagrama de flujo de los módulos del programa ejemplo 19.1


Inserción de un elemento en la cola
C
¿está la cola llena: F = I – 1 ó I = 1 y F = 25 ?
sí
no
hacer indicador de resultado S = 0
D
insertar el elemento Q(F) = A
¿está F al final de la matriz, es decir, es F = 25?
sí
no
incrementar el puntero F = F + 1
hacer el puntero F = 1
hacer el indicador de resultado S = 1
D


Extracción de un elemento de la cola
E
¿está
la cola no
vacía:
F ≠ I?
sí
no
hacer
indicador de
resultado
S = 0
F
extraer el
elemento de
la cola
B = Q(I)
¿está
I al final de
la matriz
I = 25?
sí
no
incrementar
el puntero
I = I + 1
hacer el
puntero
I = 1
hacer el
indicador de
resultado
S = 1
F

Creación de una cola vacía

```
2000 REM CREACION DE UNA COLA VACIA
2005 REM SIN PARAMETROS
2010 LET F = 1
2015 LET I = 1
2020 RETURN
```

Inserción de un elemento en la cola

```
2100 REM INTRODUCIR UN ELEMENTO EN LA COLA
2105 REM PARAMETROS
2110 REM A: ELEMENTO INTRODUCIDO
2115 REM S: INDICADOR DE ACIERTO/FALLO
2120 IF F = I - 1 OR I = 1 AND F = 25 THEN 2130
2125 GOTO 2140
2130 LET S = 0
2135 RETURN
2140 LET Q(F) = A
2145 IF F = 25 THEN 2160
2150 LET F = F + 1
2155 GOTO 2165
2160 LET F = 1
2165 LET S = 1
2170 RETURN
```

Extracción de un elemento de la cola

```
2200 REM EXTRACCION DE UN ELEMENTO DE LA COLA
2205 REM PARAMETROS
2210 REM B: ELEMENTO EXTRAIDO
2215 REM S: INDICADOR DE ACIERTO/FALLO
2220 IF F <> I THEN 2235
2225 LET S = 0
2230 RETURN
2235 LET B = Q(I)
2240 IF I = 25 THEN 2255
2245 LET I = I + 1
2250 GOTO 2260
2255 LET I = 1
2260 LET S = 1
2265 RETURN
```

Puntos de interés

- En el módulo de extracción se ha invertido la condición establecida en el algoritmo para simplificar la estructura del programa. Dada la complejidad de la instrucción condicional del módulo de inserción, no se ha empleado con ella ese recurso.

19.4

Algunas aplicaciones de las colas

Las colas se usan a veces en programas de aplicaciones, sobre todo en aplicaciones en tiempo real. De todas formas, como las pilas, se emplean sobre todo en programas del sistema. A este nivel se emplean en los sistemas operativos, para controlar un trabajo, para transferir datos de un dispositivo a otro y en comunicaciones de datos. Algunas de estas aplicaciones se investigarán más a fondo en el ejercicio propuesto al final del capítulo.

19.5

Conclusión

En este capítulo hemos visto las propiedades generales de las colas y hemos examinado un programa para llevarlas a la práctica, aunque hay que insistir en que el uso de una memoria circular no es la única forma de realizar esta estructura. En el ejercicio propuesto al final se examinará otra posibilidad.

Los puntos más importantes son:

- Una cola es una colección de datos almacenados en orden que se introducen por detrás y se extraen por delante. Es una estructura del tipo primero-en-entrar—primero-en-salir (FIFO. *First In, First Out*).
- En la mayor parte de las plasmaciones prácticas de una cola se usan dos punteros, uno para señalar el extremo inicial o anterior y otro para referirse al final o posterior.
- Casi todas las aplicaciones de las colas están al nivel del soporte lógico del sistema.

En el ejercicio propuesto a continuación se investigarán algunas de tales aplicaciones.

19

Ejercicio

1. Defina brevemente los siguientes términos: cola, FIFO, memoria circular.

2. Indique los contenidos de una cola inicialmente vacía después de ejecutar cada una de las operaciones de la siguiente secuencia:

   insertar 3168
   insertar 4267
   extraer el primer elemento
   insertar 3135
   extraer el primer elemento
   extraer el primer elemento
   insertar 3258
   extraer el primer elemento

3. Escriba un módulo adicional para el programa de manipulación de colas que indique la longitud de la cola en un momento dado.
4. Cuando se conectan dos dispositivos de un ordenador, es habitual incluir un espacio de almacenamiento llamado memoria intermedia o **tampón** en el interfaz de ambos que compensa la diferencia de velocidad entre uno y otro. Escriba un programa que simule la acción de una memoria tampón dispuesta entre dos dispositivos de comunicación con arreglo a las siguientes especificaciones:

   - La memoria tampón adoptará la forma de una cola de un tamaño máximo adecuado que contendrá una serie de **mensajes.**
   - El programa ha de responder a las órdenes siguientes:

     E enviar un mensaje.
     R recibir un mensaje.
     F fin del programa.

   - Cuando se da la orden **E**, el programa comprueba si hay espacio disponible en la cola. En caso afirmativo, solicita la introducción del texto del mensaje, que se incorpora a la cola. En caso negativo, aparece el mensaje **ESPERE.**
   - Cuando se da la orden **R**, el programa comprueba si hay algún mensaje en la cola. En caso afirmativo lo extrae y lo presenta. En caso negativo, aparece el mensaje **TAMPON VACIO**.
   - El programa se estructura en tres niveles. El superior interpreta las órdenes y presenta un mensaje adecuado si se produce alguna incorrecta. El nivel central dispone de un módulo por cada orden. Y el inferior consta de los módulos del programa ejemplo 19.1 más el escrito como respuesta a la pregunta 3. Todos estos módulos deben modificarse de manera que acepten datos alfanuméricos en la cola.
5. Escriba la documentación para el programador y para el usuario correspondiente al programa creado como respuesta a la pregunta 4.
6. Vuelva a escribir los módulos del programa ejemplo 19.1 haciendo uso de la construcción **IF ... THEN ... ELSE** y agrupando varias instrucciones en una sola línea.
7. Otra forma de crear una cola y ejecutar las operaciones asociadas a la misma es:

   - La cola se almacena en una matriz de la forma usual.
   - El extremo anterior de la cola se fija al elemento superior de la matriz.
   - Se emplea un indicador para señalar el espacio disponible tras el extremo posterior de la cola.
   - La inserción de un nuevo elemento se realiza por detrás, en la posición señalada por el indicador, que se incrementa en 1.

- Cada vez que se extrae un elemento del extremo inicial de la cola, todos los demás avanzan una posición dentro de la matriz, y el valor del indicador final se reduce en 1.

Escriba una serie de módulos equivalentes a los del programa ejemplo 19.1 para crear una cola con arreglo a las anteriores especificaciones.


RESET
RETURN

20 Listas

Este es el tercero de los capítulos dedicados al estudio de estructuras de datos fundamentales. Veremos en él una muy versátil y muy usada que se llama **lista.** Como en los dos anteriores, empezaremos por hacer una relación de sus propiedades generales, para a continuación escribir un programa que plasme en la práctica esas propiedades de la forma más sencilla posible.

En este capítulo haremos un uso intensivo del concepto de **puntero,** expuesto por vez primera en el 18. También se utilizarán mucho las técnicas de diseño, comprobación y documentación de programas estudiadas con anterioridad. Volveremos sobre el material de este capítulo en algunos de los situados más adelante.

20.1 Propiedades de las listas

En términos generales, una lista es una estructura que contiene un conjunto de datos almacenados con cierto orden. Los elementos pueden insertarse y extraerse en cualquier punto de la lista.

A consecuencia de esa forma de inserción y extracción, los elementos de la lista no pueden almacenarse en posiciones de memo-

ria consecutivas. La forma más usual de crear una lista obliga a recurrir a los punteros: se coloca uno para iniciar la lista y cada nuevo dato es acompañado por su correspondiente puntero al siguiente componente de la lista. El conjunto formado por un dato y su correspondiente puntero se llama **elemento de la lista.** El último elemento lleva un **puntero nulo,** que no señala ningún dato.

La idea general de lista se ilustra en la figura 20.1. Las operaciones de inserción y extracción se recogen en las figuras 20.2 y 20.3.

En casi todas las aplicaciones de las listas, cada uno de los elementos contiene un registro en lugar de un dato aislado. No obstante, por razones de sencillez, en la mayor parte de los ejemplos de listas utilizados aquí habrá un solo dato por elemento.

20.2

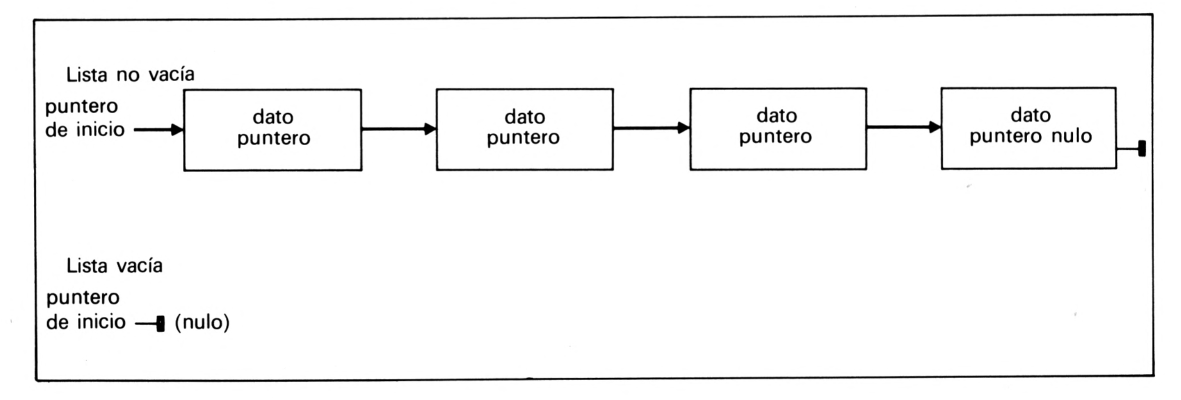
Representación de una lista en Basic

Representar una lista en Basic plantea problemas considerables. Como en el caso de las pilas y las colas, los punteros se crean por intermedio de una matriz que alberga la lista completa. Como ya sabemos, esto impone un límite al tamaño máximo de la lista almacenable.

En términos de nombres de variables, si **L** es la matriz en que está almacenada la lista y **P** es un puntero, **L(P)** es el dato contenido en el elemento de la lista. **L(P + 1)** es la porción del elemento de la lista correspondiente al puntero, y **L(L(P + 1))** es el dato del elemento siguiente. **L(L(P + 1) + 1))** es la porción correspondiente al puntero del elemento siguiente.

Otro problema es la adquisición de nuevos elementos para su inserción en la lista y el borrado de los elementos extraídos de la

Figura 20.1
Concepto general de lista



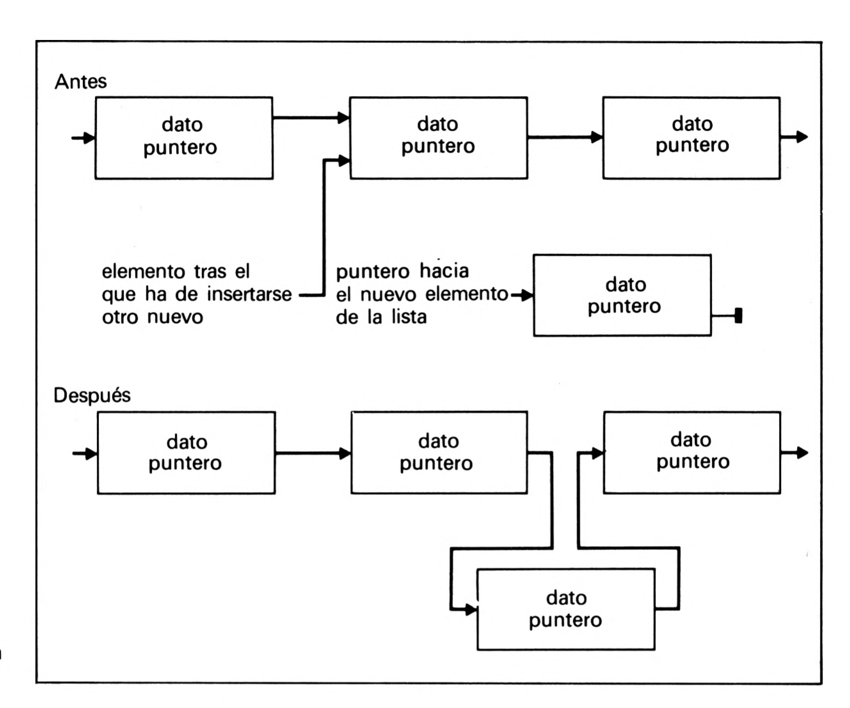


Figura 20.2
Inserción de un elemento en una lista

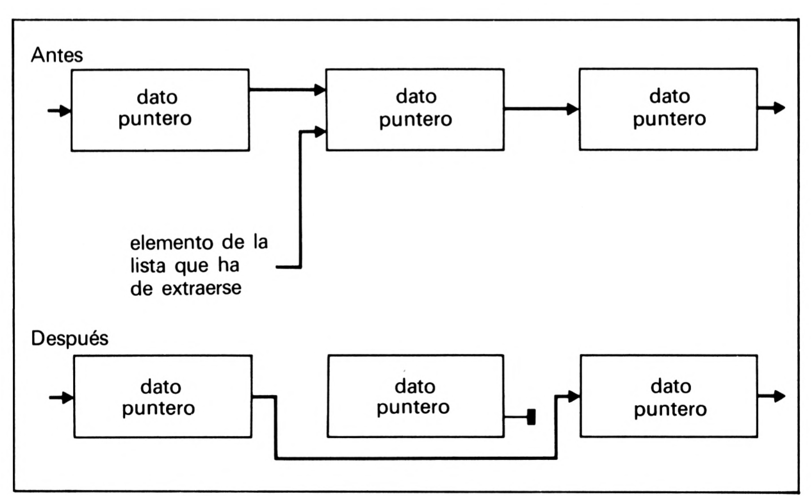


Figura 20.3
Extracción de un elemento de la lista

misma. En algunos lenguajes de programación, la inserción y el borrado de componentes es automática, pero no es ese el caso del Basic.

La solución adoptada aquí consiste en dividir la matriz que soporta la estructura en dos listas. Una de ellas almacena los datos y la otra una serie de elementos de lista vacíos. Para insertar un componente en la lista se extrae un elemento de la zona vacía, se carga con un dato y se inserta en la lista de datos. Para eliminarlo se

transfiere de esta lista a la vacía. Esta forma peculiar de crear una lista se ilustra en la figura 20.4.

20.3

Programa ejemplo 20.1

El objetivo de este programa es proporcionar una serie de módulos de servicio, encargados cada uno de ellos de ejecutar una operación sobre una lista. Las operaciones son:

1) crear una lista vacía,
2) colocar un elemento al principio de la lista.

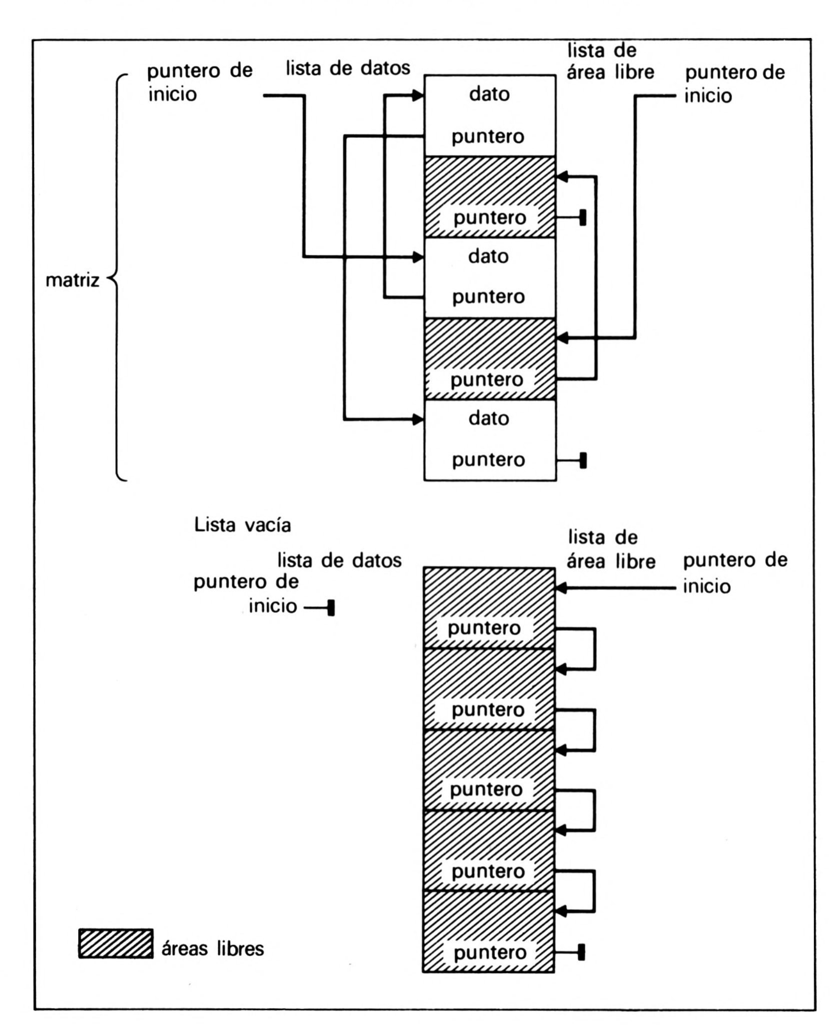
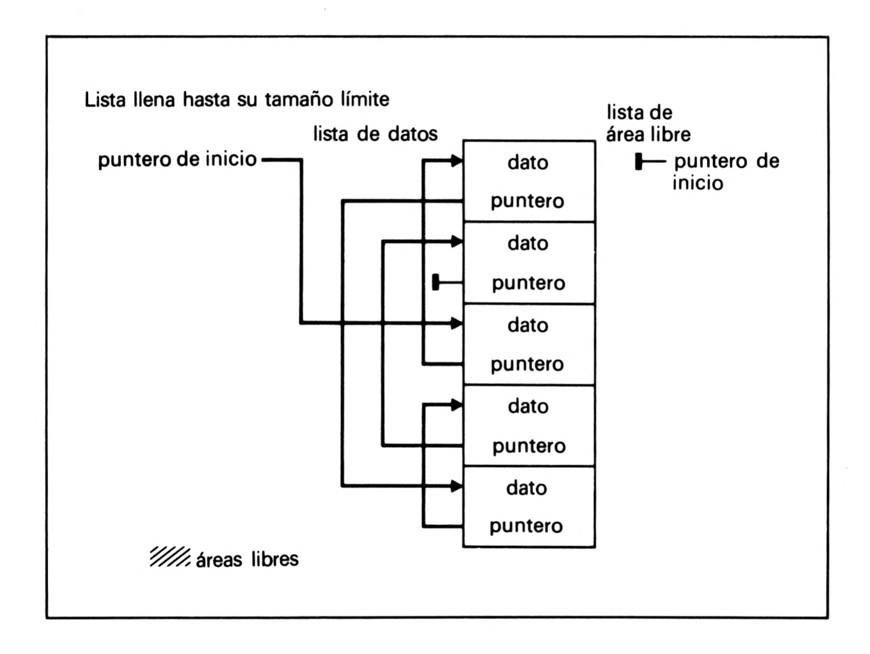


Figura 20.4
Una plasmación práctica del concepto de lista



3) localizar un elemento dentro de la lista,
4) insertar un elemento tras otro dado,
5) borrar un elemento situado tras otro dado.

Diseño del programa

El diseño general del programa es similar al utilizado en los de manipulación de pilas y colas de los ejemplos 18.1 y 19.1. Consta de cinco módulos escritos en forma de subprogramas más un programa principal no incluido aquí.

La lista se almacena en una matriz tal como se ha indicado en la sección anterior. Esta matriz, al igual que los punteros de los datos y la lista de área libre, es accesible a todos los módulos. El resto de la información pasa de unos módulos a otros en forma de parámetros.

Especificaremos ahora la actividad de cada módulo.

Creación de una lista vacía

Este módulo crea una lista vacía como la ilustrada en la figura 20.4. El puntero de partida de la lista de datos se pone a cero, lo que representará un puntero nulo a lo largo de todo este programa. El puntero de partida de la zona libre se pone a 1.

Colocación de un elemento al principio de la lista

Se consigue extrayendo un elemento al principio de la lista de área libre y colocándolo al principio de la de datos. Para ello basta modificar tres punteros, como ilustra la figura 20.5. Por razones de claridad, en esta figura se han omitido los punteros que permanecen invariables.

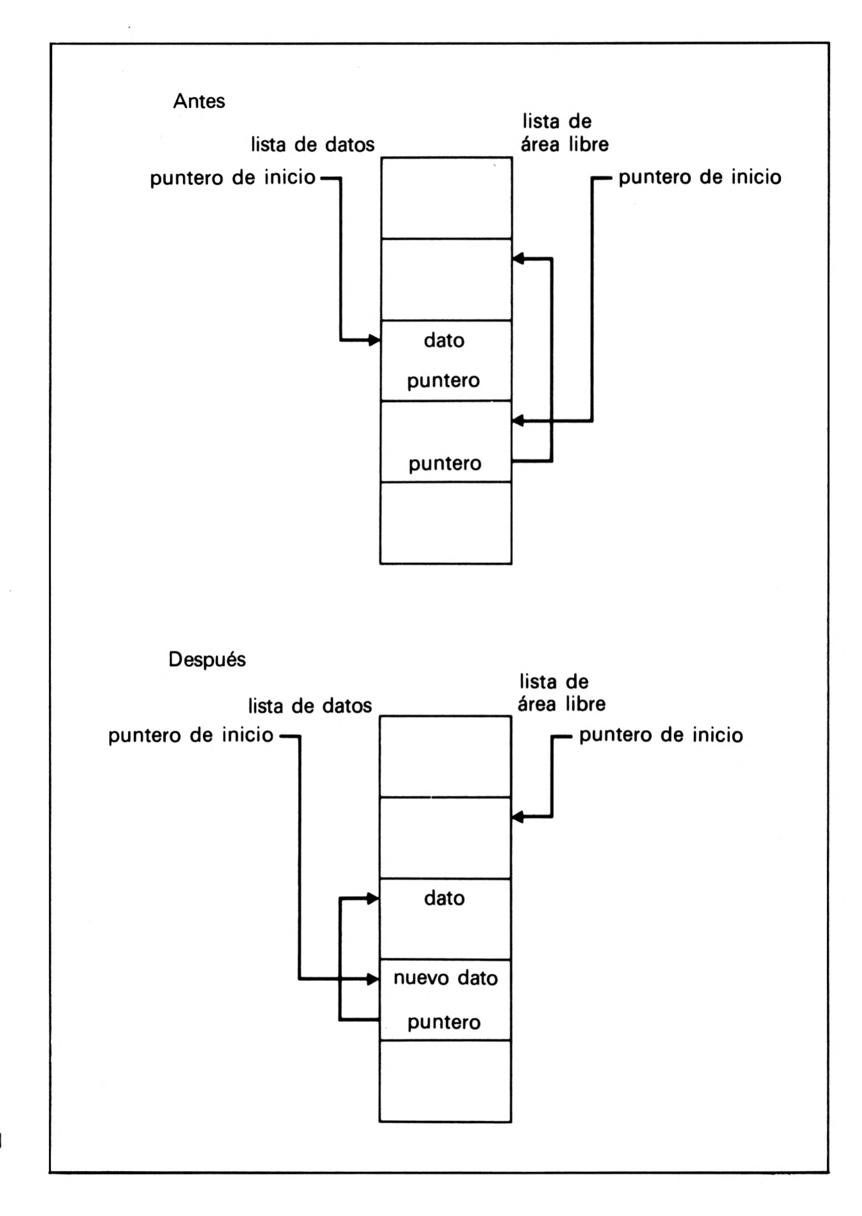


Figura 20.5
Colocación de un elemento al principio de la lista

Dado que cada puntero adopta un valor de alguno de los otros, es necesario introducir una variable provisional para almacenar un valor. Con los nombres de variables indicados a continuación,

D	Puntero de inicio de la lista de datos,
F	Puntero de inicio de la lista libre,
L(F + 1)	Puntero del segundo elemento de la lista de área libre,
T	Variable provisional,

si **F** no es cero, la secuencia de cambios es:

T	se convierte en **L(F + 1)**.
L(F + 1)	se convierte en **D**.
D	se convierte en **F**.
F	se convierte en **T**.

A continuación se inserta un dato en el nuevo elemento de la lista.

Si **F** es cero, la zona de la lista de área libre está vacía y no puede tener lugar ninguna inserción.

Localizar un elemento dentro de la lista

En este contexto, localizar un elemento significa determinar el valor de su puntero dado el dato almacenado en dicho elemento. Este módulo funciona como sigue:

Llevar el puntero al principio de la lista.

Mientras el puntero no sea nulo o
no señale el dato deseado,

Repetir el proceso:

Hacer que el puntero señale el siguiente elemento de la lista.

Si la repetición se detiene en la primera condición, es que el dato buscado no está en la lista. Si se detiene en la segunda, es que se ha encontrado y el puntero señala la posición que ocupa. Los parámetros de este subprograma son el dato buscado y el puntero.

Insertar un elemento tras otro dado

Los parámetros de este módulo son un puntero para el elemento dado y el dato que ha de almacenarse en el nuevo elemento. Funciona

extrayendo un elemento de la lista de la zona libre e insertándolo tras el elemento dado. Para ello hay que modificar varios punteros, tal como ilustra la figura 20.6. Por razones de claridad, los punteros no afectados por el cambio se han omitido. También en esta ocasión cada puntero asume el valor de uno de los otros, por lo que es preciso crear una variable provisional para almacenar un valor. Con los nombres de variables indicados a continuación,

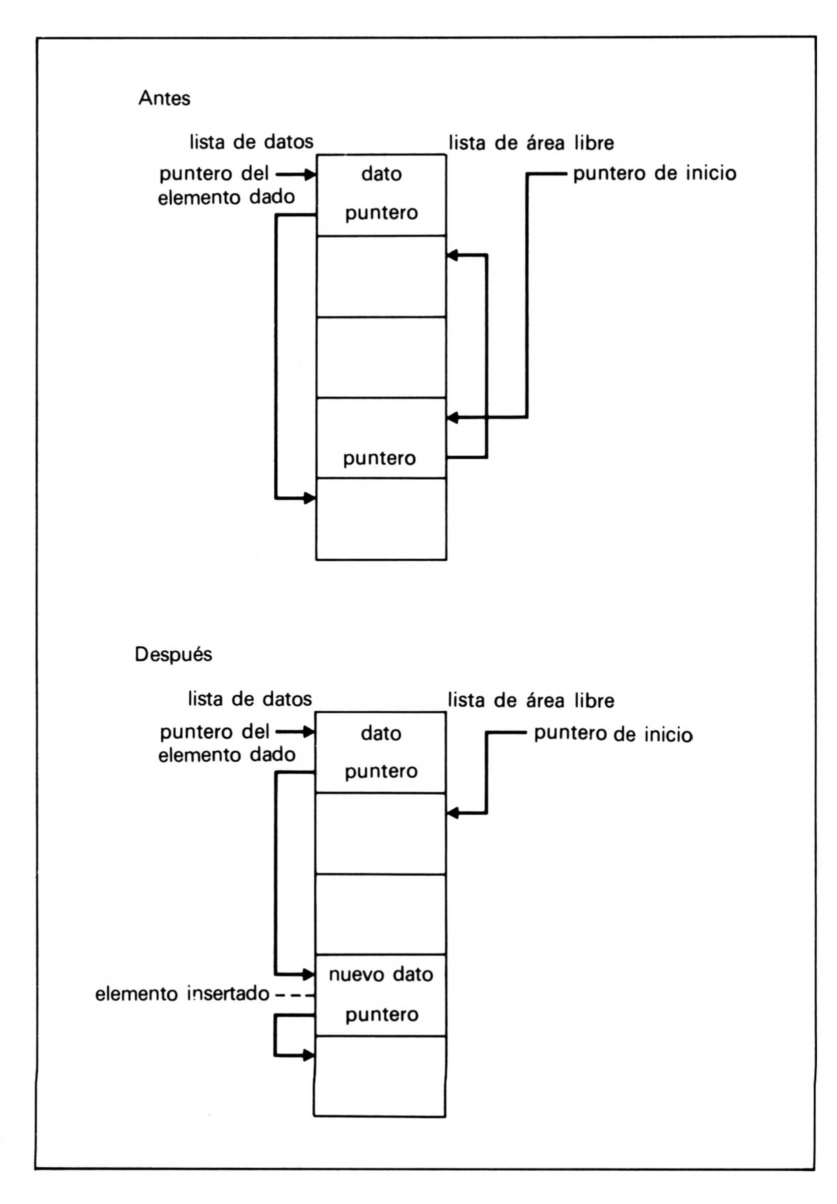


Figura 20.6
Inserción de un elemento tras otro dado

P	Puntero del elemento dado,
F	Puntero de inicio de la lista de área libre,
L(F + 1)	Puntero del segundo elemento de la lista de área libre,
T	Variable provisional,
L(P + 1)	Puntero del elemento situado tras el dado,

si **F** no es cero, la secuencia de cambios es:

T	se convierte en **L(F + 1).**
L(F + 1)	se convierte en **L(P + 1).**
L(P + 1)	se convierte en **F**.
F	se convierte en **T**.

A continuación se inserta un dato en el nuevo elemento de la lista.

Si **F** es cero, la lista de la zona libre está vacía y no puede tener lugar ninguna inserción.

Extracción de un elemento situado tras otro dado

El parámetro para este módulo es un puntero para el elemento de la lista situado delante del que ha de borrarse. El elemento que ha de borrarse se aparta de la lista de datos y se lleva a la lista libre. Para ello es preciso modificar varios punteros, como ilustra la figura 20.7. Por razones de sencillez, no se han representado los punteros que permanecen invariables. Dados los nombres de variables siguientes,

P	Puntero del elemento dado,
F	Puntero de partida de la lista libre,
L(P + 1)	Puntero del elemento que ha de extraerse,
L(L(P + 1) + 1)	Puntero del elemento que ha de extraerse,
T	Variable provisional,

si **L(P + 1)** no es cero, la secuencia de cambios es:

T	se convierte en **L(P + 1)**.
L(P + 1)	se convierte en **L(L(P + 1) + 1)** o **L(T + 1)**.
L(T + 1)	se convierte en **F**.
F	se convierte en **T**.

Si **L(P + 1)** es cero, es que no hay ningún elemento tras el dado y la extracción no puede efectuarse.

Los módulos ya están suficientemente detallados y puede procederse a escribir las rutinas del programa.

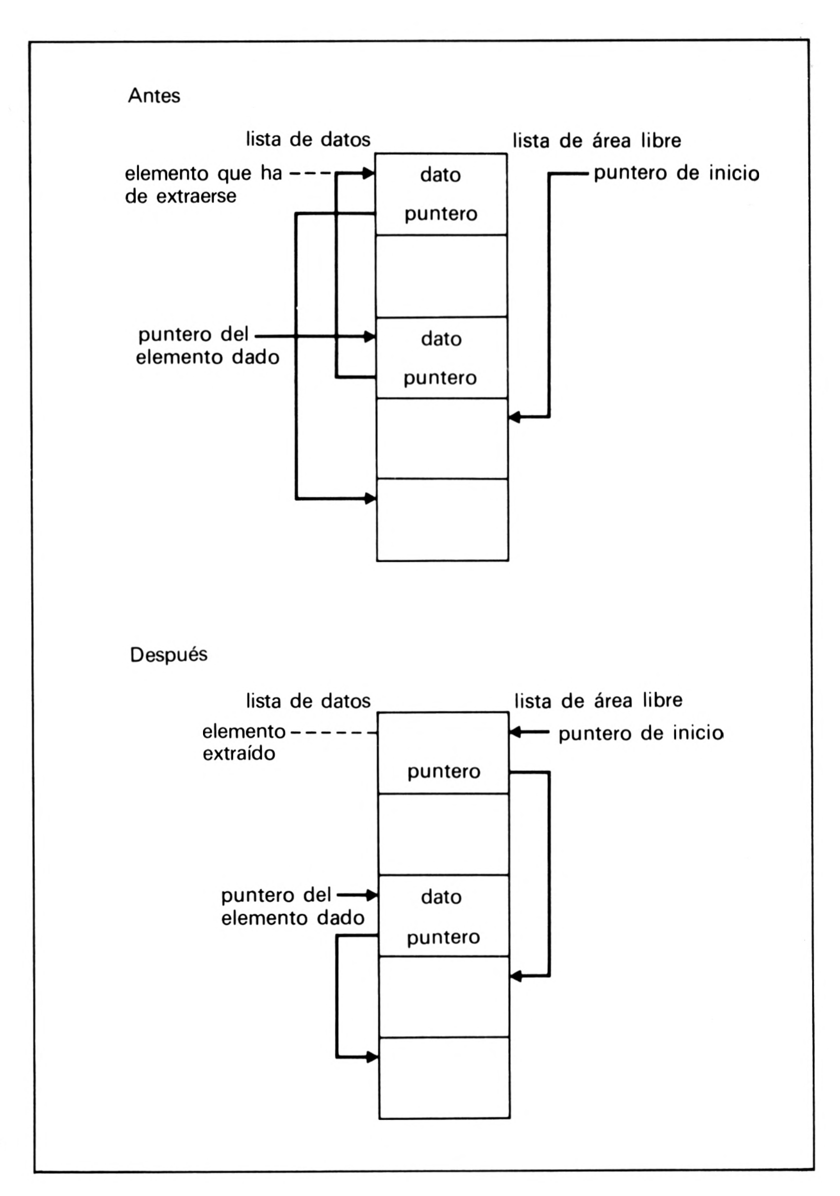


Figura 20.7
Extracción de un elemento situado detrás de otro dado

Variables

L(30)	Matriz para almacenar la lista.
D	Puntero de inicio de la lista.
F	Puntero de inicio de la lista de área libre.
K	Contador de bucle.
T	Variable provisional.

P Puntero de un elemento particular de la lista.
I Dato almacenado en un elemento particular de la lista.
S Parámetro de resultado: **S** = 1 si la operación en curso se ha realizado con éxito; **S** = 0 en caso contrario.

Diagrama de flujo

La figura 20.8 ilustra el flujo de control de los módulos del programa ejemplo 20.1.

Programa

```
1000 REM PROGRAMA EJEMPLO 20.1
1005 REM MANIPULACION DE LISTA
1010 REM VARIABLES GLOBALES
1015 REM L: MATRIZ PARA ALMACENAR LA LISTA
1020 REM D: PUNTERO AL INICIO DE LA LISTA
1025 REM F: PUNTERO AL INICIO DE LA ZONA LIBRE
1030 DIM L(30)
1035 REM
```

Creación de una lista vacía

```
2000 REM CREACION DE UNA LISTA VACIA
2005 LET D = 0
2010 LET F = 1
2015 FOR K = 2 TO 28 STEP 2
2020 LET L(K) = K + 1
2025 NEXT K
2030 LET L(30) = 0
2035 RETURN
```

Colocación de un elemento al principio de la lista

```
2100 REM COLOCACION DE UN ELEMENTO AL PRINCIPIO
         DE LA LISTA
2105 REM PARAMETROS
2110 REM I: DATO ALMACENADO EN EL NUEVO ELEMENTO
2115 REM S: INDICADOR DE ACIERTO/FALLO
2120 IF F <> 0 THEN 2135
2125 LET S = 0
2130 RETURN
2135 LET T = L(F + 1)
2140 LET L(F + 1) = D
2145 LET D = F
2150 LET F = T
2155 LET L(D) = I
2160 LET S = 1
2165 RETURN
```

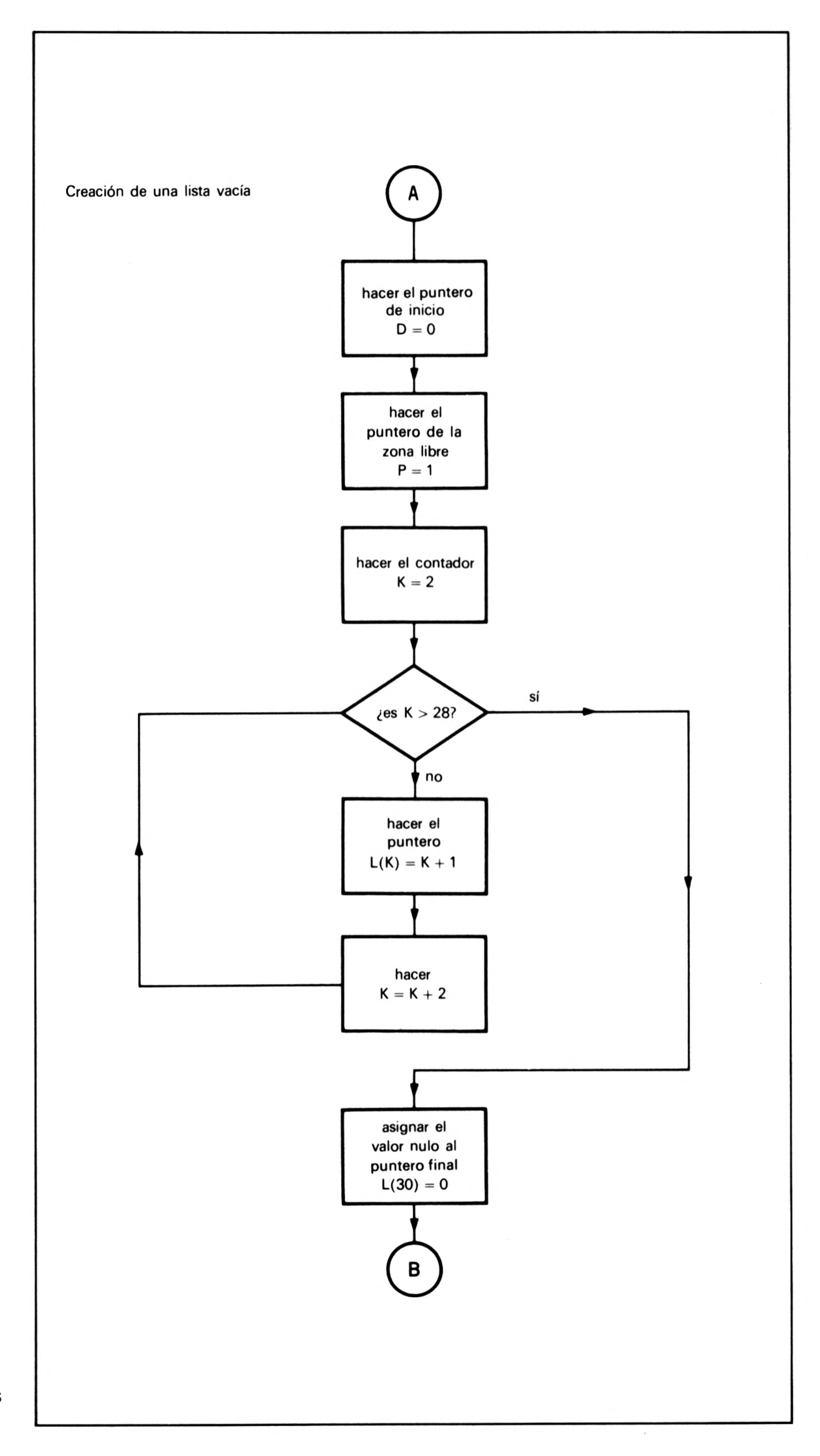
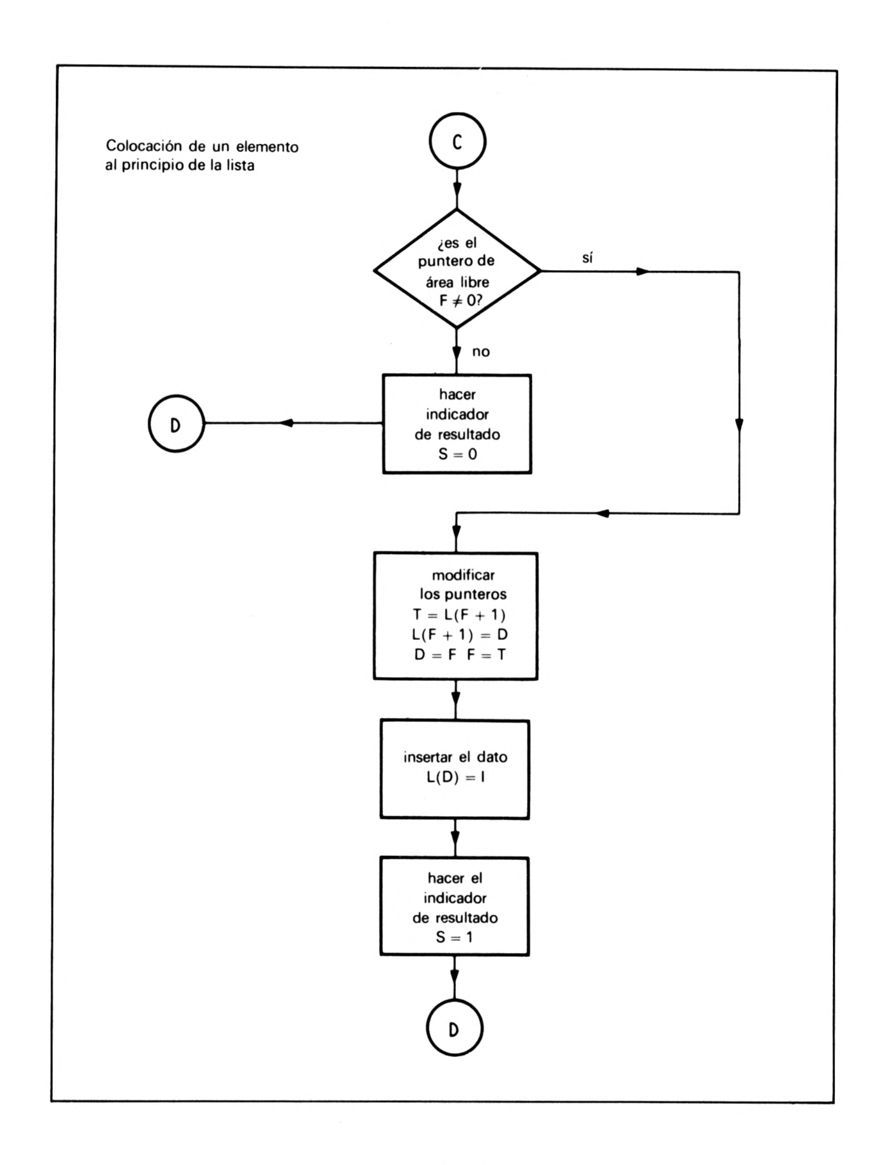


Figura 20.8
Diagrama de flujo de los módulos del programa ejemplo 20.1



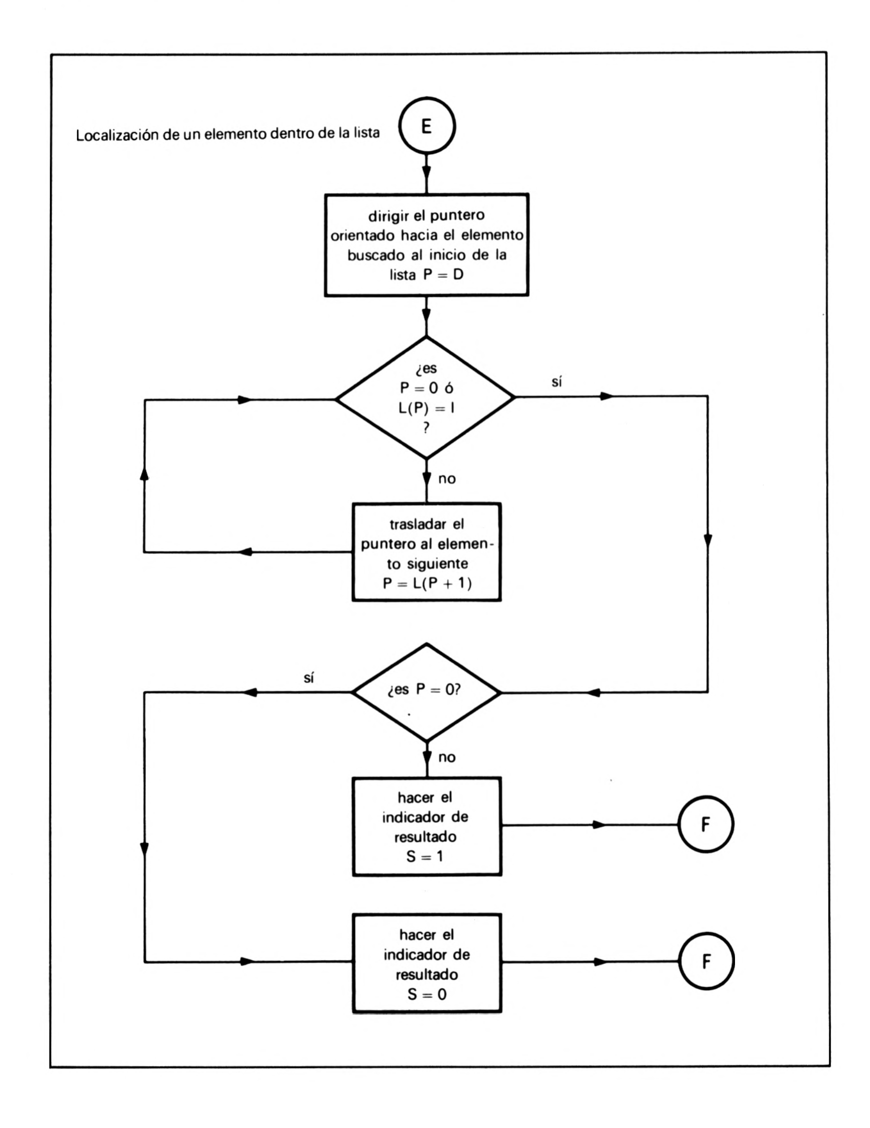
Localización de un elemento dentro de la lista

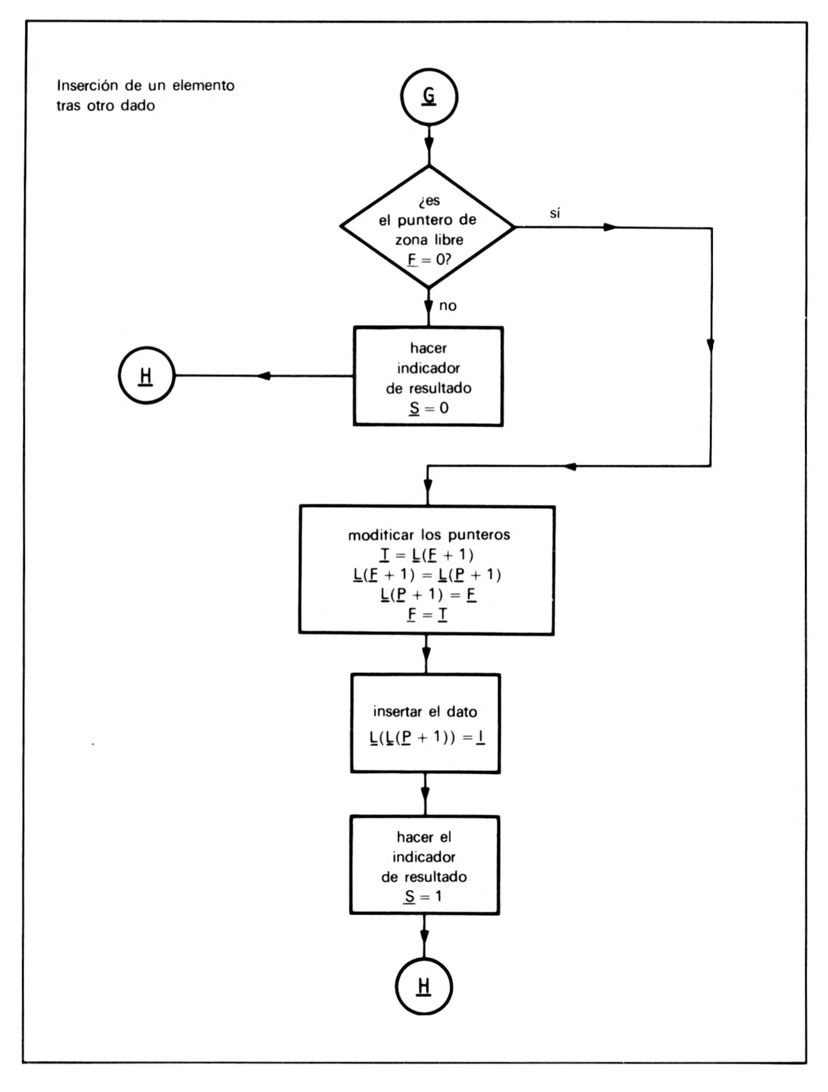
```
2200 REM LOCALIZACION DE UN ELEMENTO DENTRO DE
         UNA LISTA
2205 REM PARAMETROS
2210 REM I: DATO EN UN ELEMENTO DE LA LISTA
2215 REM P: PUNTERO DEL ELEMENTO EN LA LISTA
2220 REM S: INDICADOR DE ACIERTO/FALLO
2225 LET P = D
2230 IF P = 0 OR L(P) = I THEN 2245
2235 LET P = L(P + 1)
2240 GOTO 2230
```

```
2245 IF P = 0 THEN 2260
2250 LET S = 1
2255 RETURN
2260 LET S = 0
2265 RETURN
```

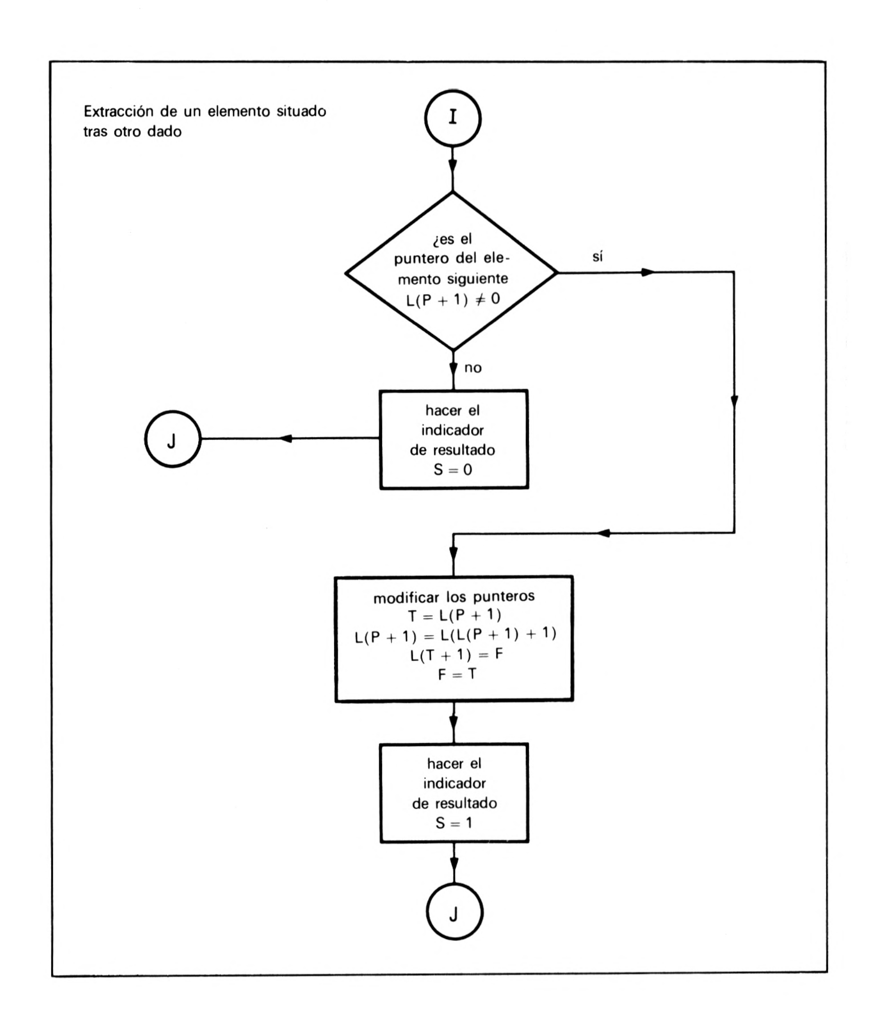
Inserción de un elemento situado tras otro dado

```
2300 REM INSERCION DE UN ELEMENTO TRAS OTRO DADO
         DE UNA LISTA
2305 REM PARAMETROS
2310 REM I: DATO ALAMACENADO EN EL NUEVO ELEMENTO
```






```
2315 REM P: PUNTERO DE UN ELEMENTO DADO DE LA
            LISTA
2320 REM S: INDICADOR DE ACIERTO/FALLO
2325 IF F <> 0 THEN 2340
2330 LET S = 0
2335 RETURN
2340 LET T = L(F + 1)
2345 LET L(F + 1)= L(P + 1)
2350 LET L(P  + 1) = F
2355 LET F = T
2360 LET L(L(P + 1)) = I
2365 LET S = 1
2370 RETURN
```



Extracción de un elemento situado tras otro dado

```
2400 REM EXTRACCION DE UN ELEMENTO SITUADO TRAS
         OTRO DADO DE UNA LISTA
2405 REM PARAMETROS
2410 REM P: PUNTERO DE UN ELEMENTO DADO DE LA
            LISTA
2415 REM S: INDICADOR DE ACIERTO/FALLO
2420 IF L(P + 1) <> 0 THEN 2435
2425 LET S = 0
2430 RETURN
2435 LET T = L(P + 1)
2440 LET L(P + 1) = L(T + 1)
2445 LET L(T + 1) = F
2450 LET F = T
2455 LET S = 1
2460 RETURN
```

Puntos de interés

- En el módulo que localiza un elemento dentro de la lista se ha invertido la condición de terminación del bucle para simplificar el programa.
- Para la complejidad de las tareas que llevan a cabo, estos módulos son sorprendentemente sencillos.

20.4
Algunas aplicaciones de las listas

Por su sencillez y versatilidad, la lista es una de las estructuras de datos más usadas. En programación, suelen emplearse para archivar ficheros en los que cada elemento de la lista contiene un registro y un indicador que señala el registro siguiente. Esta estructura puede también utilizarse cuando los ficheros están en una memoria auxiliar; en este caso, el indicador contiene la dirección en la memoria auxiliar del registro siguiente.

También se emplean las listas a nivel de sistema para almacenar colas de los programas que han de ejecutarse y para compartimentar la memoria principal del ordenador. Algunas de estas aplicaciones se examinarán en el ejercicio propuesto al final del capítulo.

20.5
Conclusión

Hemos estudiado en este capítulo el concepto de lista y examinado algunas de las operaciones que pueden realizarse sobre esa estructura. También se han visto algunos métodos de ejecutar esas operaciones en Basic. Los puntos más importantes son:

- Una lista es una estructura que contiene un conjunto de datos almacenados en cierto orden.
- Los datos pueden insertarse y extraerse en cualquier punto de la lista.
- La realización práctica más común de una lista consta de una serie de elementos cada uno de los cuales contiene un dato y un puntero que señala el elemento siguiente. Hay además un puntero que marca el comienzo de la lista y un puntero nulo para el último elemento.

Ejercicio

1. Defina brevemente los siguientes términos: lista, elemento de la lista, puntero nulo.
2. Copie las tres listas de la figura 20.4, numere en cada caso los elementos de la matriz a partir de 1 y rellene los valores de todos los punteros representados.
3. Copie las dos listas de la figura 20.5, numere en cada caso los elementos de la matriz a partir de 1 y rellene los valores de todos los punteros representados. Utilice las cifras así obtenidas para realizar un pase de prueba con el módulo del programa que efectúa la operación equivalente.
4. Repita las operaciones de la pregunta 3 para las figuras 20.6 y 20.7.
5. Escriba un módulo adicional de manipulación de listas que borre el primer elemento de una lista.
6. El concepto de lista expuesto en este capítulo puede utilizarse para llevar a la práctica el de cola propuesto en el anterior.

   Vuelva a escribir los módulos del programa ejemplo 19.1 en términos de la estructura de datos descrita en la sección 20.2 y en los módulos del programa ejemplo 20.1 y de la pregunta 5 de este ejercicio.
7. Diseñe y escriba un programa para mantener una lista actualizada de números de cuenta almacenados en orden numérico ascendente.

   - El programa responde a las siguientes órdenes:

INSERTAR	seguido del número de cuenta que ha de insertarse en la posición adecuada.
BORRAR	seguido del número de cuenta que ha de borrarse.
LISTA	presenta a la salida la lista ordenada.
FIN	interrumpe el programa.

   - El programa se estructura en tres niveles. En el superior hay un módulo que introduce e identifica las órdenes y transfiere el control al módulo encargado de llevar a cabo la orden en cuestión. Las órdenes no reconocidas se rechazan.
   - En el segundo nivel hay un módulo por cada una de las órdenes más otro de iniciación. Estos se encargan, en caso necesario, de introducir la información asociada a una orden y llama a varios módulos de servicio para que la ejecuten.
   - El tercer nivel está constituido por varios módulos de servicio, que incluyen los de la pregunta 5 y el programa ejemplo 20.1, con excepción del encargado de localizar un elemento dentro de la lista. A continuación se describirá una versión modificada de este último.
   - Para insertar un número de cuenta se recorre la lista hasta dar con un elemento que contenga un número de cuenta mayor: el nuevo número se inserta delante de aquél. Como las inserciones sólo pueden efectuarse detrás de un elemento dado, es preciso crear un módulo que recorra la lista con dos punteros, **P1** y **P2**, dirigido el primero al elemento en curso y retrasado el segundo en un elemento, de forma tal que quede señalando al elemento anterior.

   En este módulo pueden producirse varias situaciones diferentes:

   P1 y **P2** son diferentes de cero: **P1** señala el primer elemento cuyo número de cuenta es mayor o igual que el del elemento dado; por tanto, **P2** señala el elemento tras el cual hay que insertar el nuevo.

P1 no es cero y **P2** es cero: el nuevo elemento ha de insertarse al principio de la lista.

P1 es cero y **P2** no: el nuevo elemento ha de insertarse al final de la lista, tras el señalado por **P2**.

P1 y **P2** son cero: la lista está vacía. El nuevo elemento debe insertarse al inicio de la misma.

El módulo del segundo nivel encargado de ejecutar la orden **INSERTAR** llama a este módulo de barrido y, según los valores de los punteros que le son entregados, llama a uno u otro de los dos módulos de inserción.

El módulo de barrido descrito se usa también cuando hay que extraer un número de cuenta. Entrega los valores de los punteros, tal como acaba de decirse.

Si **P1** no es cero y el elemento al que señala es el que hay que extraer, **P2** señalará al elemento anterior, y la extracción podrá llevarse a cabo. Si **P2** es cero, el elemento ha de extraerse de la cabecera de la lista. Si **P1** es cero o no señala el elemento deseado, es que éste no está en la lista.

Es preciso escribir un módulo de servicio adicional para mostrar los elementos de la lista.

8. Otra forma de llevar a la práctica el concepto de lista es utilizar punteros de avance y retroceso, como los ilustrados en la figura 20.9.

a) Vuelva a escribir los módulos del programa ejemplo 20.1 y de la pregunta 5 a la luz de este nuevo procedimiento.

b) Vuelva a escribir el programa de la pregunta 6 de esta misma manera.

Figura 20.9
Lista con punteros de avance y retroceso

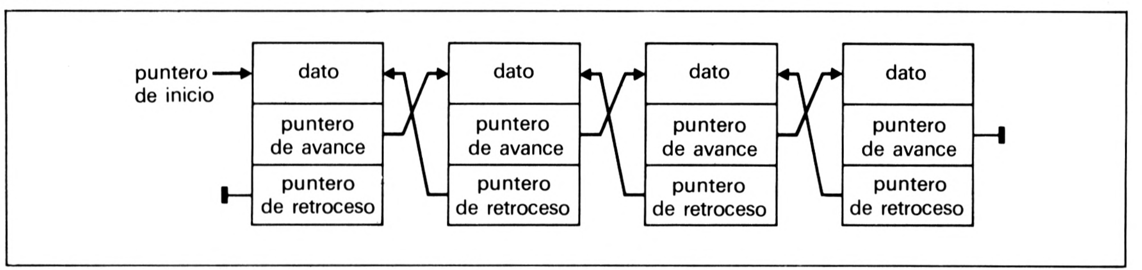
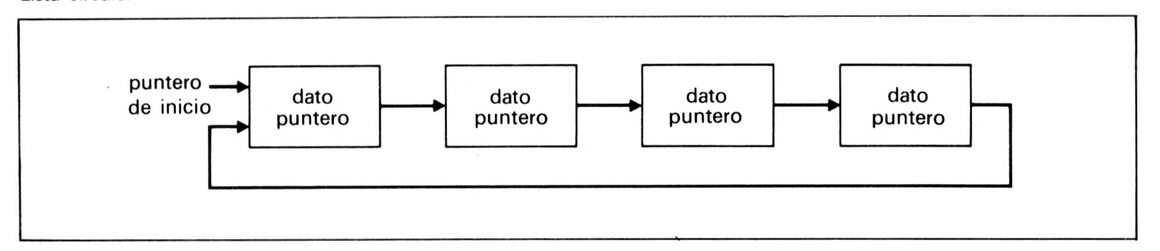


Figura 20.10
Lista circular



9. Redacte la documentación del programador y la guía del usuario correspondientes al programa de la pregunta 6.

10. Hay una variante de la idea de lista llamada **lista circular,** que consiste en que el puntero del elemento posterior señala al de cabeza de lista, tal como se ilustra en la figura 20.10.

Una lista circular con dos punteros sirve para realizar una cola y las operaciones asociadas a ella según lo descrito en el capítulo 19.

a) Vuelva a escribir los módulos del programa ejemplo 19.1 y de la pregunta 3 del ejercicio 19 utilizando una lista circular como estructura de datos. Utilice en la lista un número de elementos fijo. En lugar de insertar y extraer elementos como en una cola, limítese a trasladar los punteros anterior y posterior.

b) Vuelva a escribir el programa de la pregunta 4 del ejercicio 19 utilizando los módulos de la parte *a*) de esta misma pregunta.

11. Añada al programa de procesado de listas los módulos necesarios para cargar una lista de un fichero de datos archivado en memoria auxiliar en una matriz contenida en la memoria central y para copiar una lista contenida en una matriz de esta última en un archivo de datos de la auxiliar.

12. Vuelva a escribir el programa ejemplo 11.2 usando una lista como estructura de datos básica. No es preciso introducir ningún cambio en el módulo de control, pero hay que hacer algunas modificaciones leves en los de operación y escribir un nuevo juego de módulos de servicio.

Compare las dos versiones del programa en términos de sencillez, espacio de memoria ocupado y velocidad de trabajo.

13. Vuelva a escribir los módulos del programa ejemplo 20.1 utilizando la construcción **IF ... THEN ... ELSE** y agrupando varias instrucciones en una sola línea.


RESET
RETURN

21 Arboles

Este es el cuarto y último de los capítulos dedicados al estudio de estructuras de datos fundamentales. En él examinaremos el **árbol**, una estructura muy utilizada, sobre todo en los programas de compilación de lenguajes. Como en los tres capítulos anteriores, veremos primero las propiedades del árbol para a continuación diseñar y escribir un programa que las lleve a la práctica de la forma más sencilla posible. Este programa adoptará la forma de una serie de módulos de servicio utilizados por rutinas de nivel más elevado, como las escritas para satisfacer las especificaciones contenidas en el ejercicio propuesto al final del capítulo.

Se utilizará mucho el concepto de **puntero,** expuesto por vez primera en el capítulo 18. También se emplearán continuamente las técnicas de diseño, comprobación y documentación de programas aprendidas anteriormente. Por otra parte, el material presentado aquí volverá a emplearse hacia el final, concretamente en el capítulo 28, dedicado al análisis sintáctico.

21.1 Propiedades del árbol

Un árbol es una estructura de datos jerarquizada en la que cada elemento se relaciona con todos los situados bajo él. La figura 21.1 ilustra el concepto general de árbol.

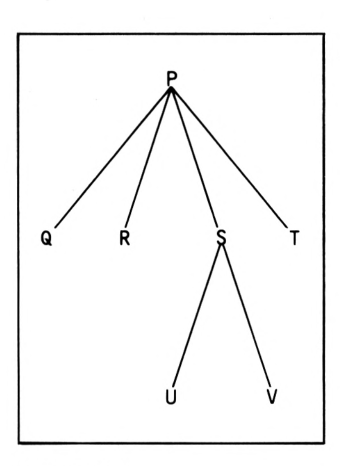


Figura 21.1
Arbol

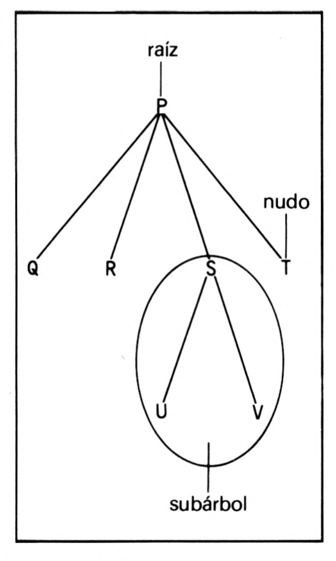


Figura 21.2
Nudos y subárboles

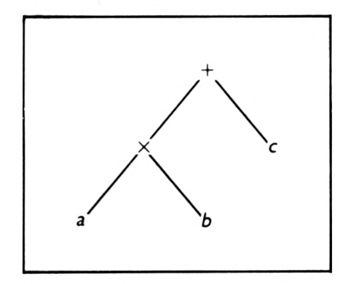


Figura 21.3
Arbol binario

Cada uno de los elementos del árbol se llama **nudo**, y cada nudo está unido a varios **subárboles.** El situado en la parte superior de la estructura se llama **raíz** del árbol. Todos estos elementos se ilustran en la figura 21.2. Una de las propiedades más importantes del árbol es que cada uno de sus subárboles es a su vez un árbol.

Se usa mucho un tipo particular de árbol en el que cada nudo tiene como máximo dos subárboles. Tal estructura, ilustrada en la figura 21.3, se llama **árbol binario.** El resto del capítulo se limitará al estudio de los árboles binarios.

Sobre un árbol pueden realizarse varias operaciones: dos árboles binarios pueden **empalmarse** por medio de una raíz adicional para formar otro árbol binario de mayor tamaño. E inversamente: dentro de un árbol pueden identificarse dos **subárboles izquierdo** y **derecho.** El dato situado en la raíz puede **examinarse.** Y, por último, un árbol puede **recorrerse** de varias formas, entendiendo por recorrer "visitar" de forma sistemática cada uno de los nudos.

Los módulos del programa estudiado en este capítulo realizarán todas las operaciones anteriores con excepción del recorrido, que está fuera del alcance de este libro.

21.2

Representación de un árbol en Basic

Como ocurre con las pilas, las listas y las colas, la mejor forma de materializar un árbol en Basic es recurrir a una matriz. Cada nudo exige el uso de tres elementos de la matriz, uno para el dato situado en ese nudo y dos para los indicadores dirigidos hacia los subárboles izquierdo y derecho.

Como en casos anteriores, la necesidad de recurrir a una matriz impone un límite al tamaño del árbol que puede almacenarse. Además, es preciso organizar en la matriz una zona libre de manera que puedan extraerse nudos de ella para insertarlos en los árboles. Por razones de comodidad, esa zona libre se organiza en forma de lista y los indicadores del subárbol derecho se usan para unir los elementos.

Esta realización práctica del concepto de árbol se ilustra en la figura 21.4, que recoge la forma en que puede almacenarse el árbol de la figura 21.3.

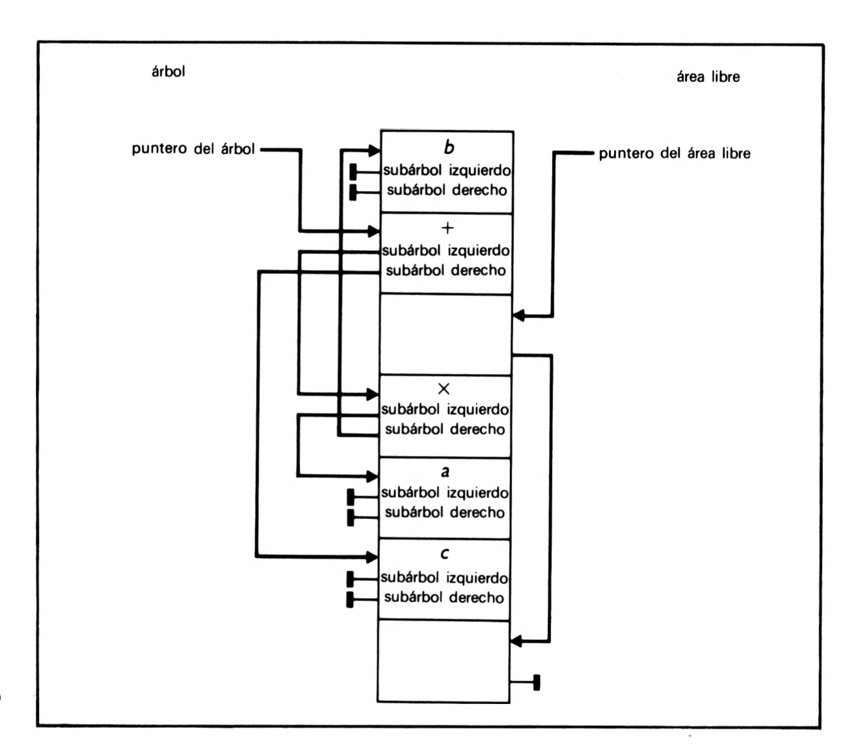


Figura 21.4
Plasmación práctica del concepto de árbol

21.3
Programa ejemplo 21.1

El objetivo de este programa es proporcionar una serie de módulos de servicio encargados cada uno de ejecutar una operación sobre un árbol. Estas operaciones son:

1) inicializar la estructura que servirá para almacenar uno o más árboles;
2) crear un árbol vacío;
3) devolver un puntero al subárbol izquierdo de un árbol dado;
4) devolver el valor de un dato situado en la raíz del árbol;
5) empalmar dos árboles por medio de una raíz adicional, insertando en la misma un dato dado.

Diseño del programa

El programa se estructura en una serie de módulos, uno por cada una de las operaciones que acaban de indicarse, escritos en forma de

subprogramas. Los módulos operan sobre ciertas variables globales, como la matriz que alberga los árboles. Los datos específicos de una operación pasan de un módulo a otro en forma de parámetros. Un árbol se identifica por medio de un puntero orientado hacia su raíz. Especificaremos ahora con cierto detalle la actividad de cada uno de los módulos.

Inicializar la estructura que servirá para almacenar uno o más árboles

Este módulo crea una matriz como la ilustrada en la figura 21.5. Como el árbol ha de almacenar datos alfanuméricos, los punteros se almacenan en forma de carácter y se transforman en sus equivalentes numéricos siempre que es necesario; dichos indicadores se transfieren y restituyen como parámetros numéricos.

Crear un árbol vacío

Este módulo entrega un puntero nulo o de valor cero, que representa un árbol vacío.

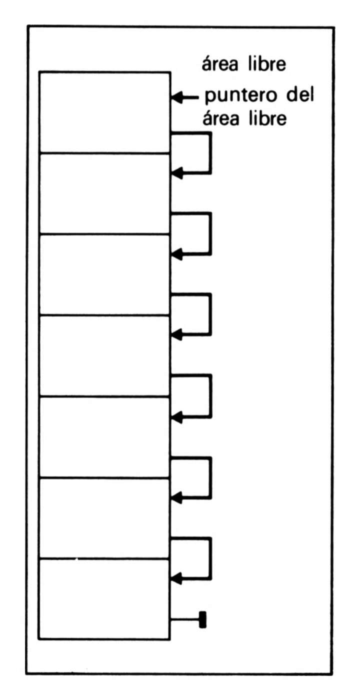


Figura 21.5
Estado inicial de la estructura de almacenamiento del árbol

Entregar un puntero al subárbol izquierdo de un árbol dado

Dado un puntero para la raíz de un árbol, el módulo entrega el valor del puntero del subárbol izquierdo de aquélla. Si el puntero dado es nulo, entrega una señal de fallo.

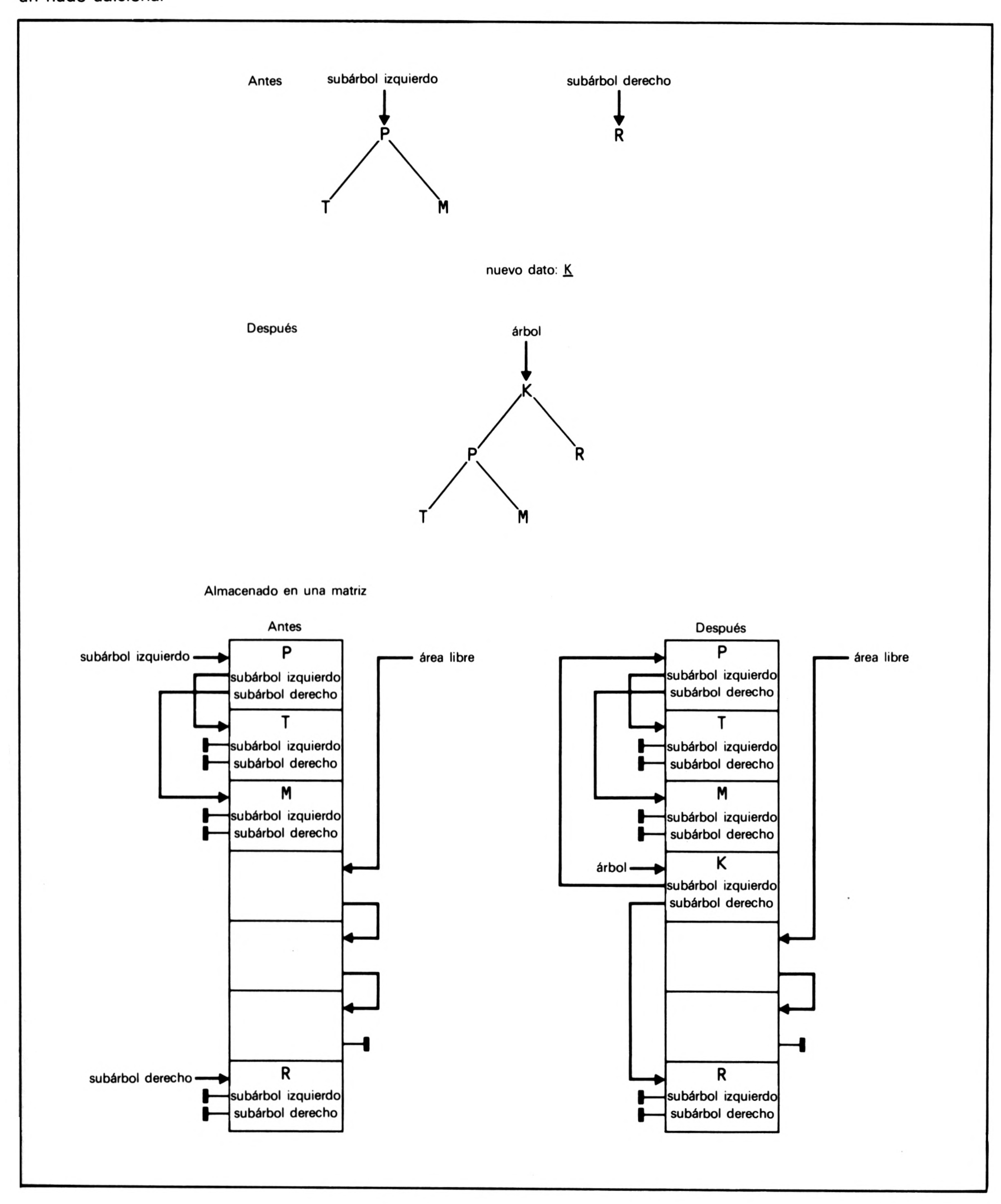
Entregar el valor del dato situado en la raíz del árbol

Dado un puntero para la raíz de un árbol, el módulo entrega el valor del dato situado en la misma. Si el puntero es nulo, entrega una señal de fallo.

Empalmar dos árboles por medio de una raíz adicional

Dados los punteros correspondientes a los dos árboles que han de empalmarse y el valor del dato de la nueva raíz, se recorre la siguiente rutina:

Figura 21.6
Empalme de dos árboles mediante un nudo adicional



Obtener un nudo en la zona libre y modificar el puntero de dicha zona. Si no hay ningún nudo libre, se genera una señal de fallo.

Insertar los punteros de los árboles a empalmar en los punteros de los subárboles izquierdo y derecho del nuevo nudo.

Insertar el dato dado en el nuevo nudo.

Devolver el puntero del nuevo nudo, que se convierte así en raíz del árbol resultante del empalme.

El proceso se ilustra en la figura 21.6. Los módulos están ya especificados con detalle suficiente y puede pasarse a la fase de escritura de las rutinas del programa.

Variables

Variables globales:

A$ Matriz para almacenar los árboles.
L Puntero del área libre.

Parámetros:

A Puntero de un árbol.
I, D Punteros de los subárboles izquierdo y derecho.
N$ Dato de la raíz.
S Indicador de resultado: **S** = 1 si la operación se ha llevado a cabo con éxito; **S** = 0 en caso contrario.

Variables locales:

K Contador de bucle.

Diagrama de flujo

La figura 21.7 ilustra el flujo de control de los módulos del programa ejemplo 21.1.

Programa

```
1000 REM PROGRAMA EJEMPLO 21.1
1005 REM MANIPULACION DE UN ARBOL
1010 REM VARIABLES GLOBALES
1015 REM A$: MATRIZ PARA ALMACENAMIENTO DE ARBO
          LES
1020 REM L: PUNTERO DE LA ZONA LIBRE
1025 DIM A$ (30)
1030 REM
```

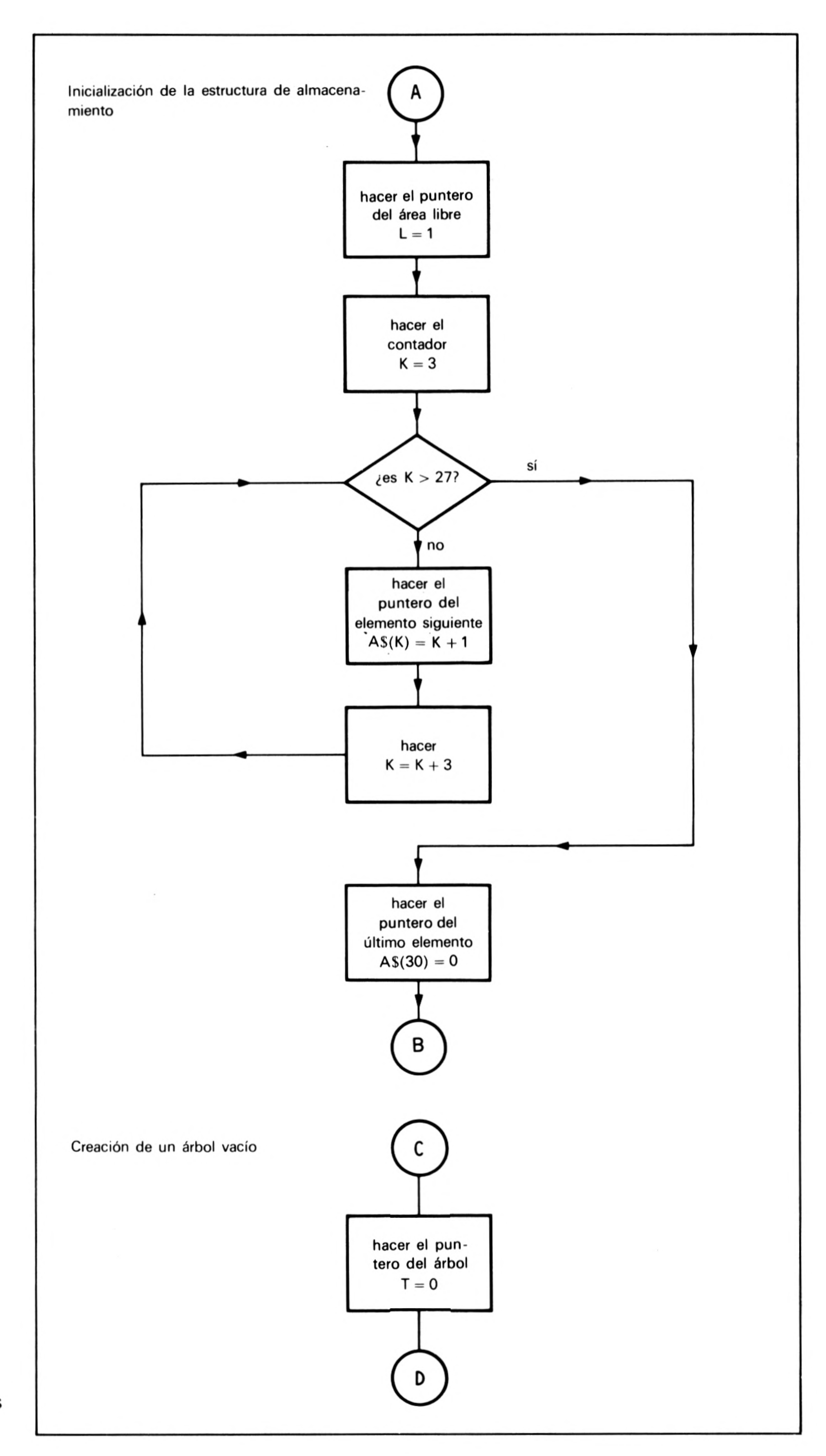
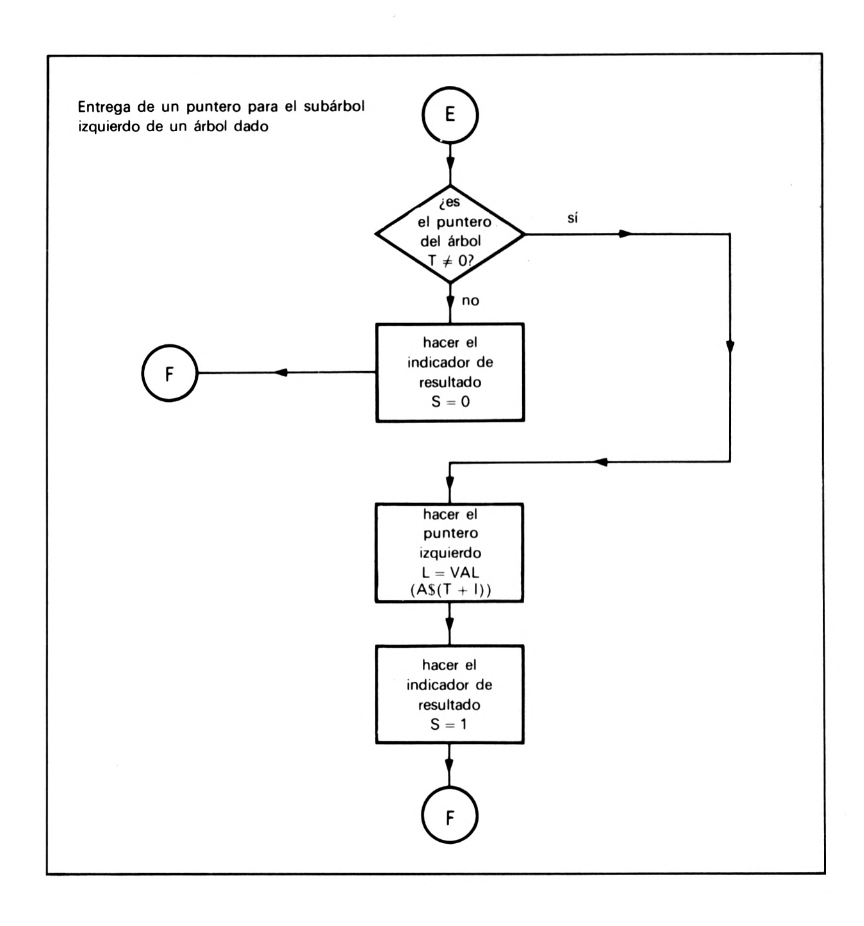


Figura 21.7
Diagrama de flujo de los módulos del programa ejemplo 21.1

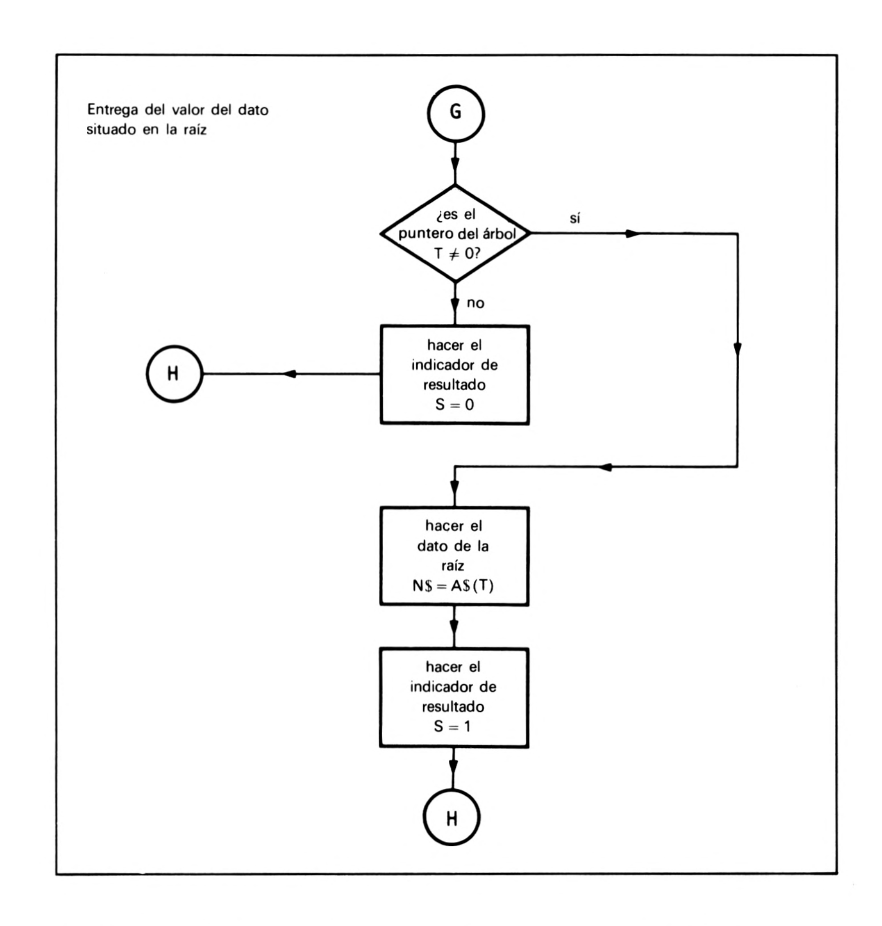


Inicializar la estructura de almacenamiento

```
2000 REM INICIALIZACION DE LA ESTRUCTURA DE ALMA
         CENAMIENTO
2005 LET L = 1
2010 FOR K = 3 TO 27 STEP 3
2015 LET A$(K) = STR$(K + 1)
2020 NEXT K
2025 LET A$(30) = "0"
2030 RETURN
```

Crear un árbol vacío

```
2100 REM CREACION DE UN ARBOL VACIO
2105 REM PARAMETROS
2110 REM A: PUNTERO DE ARBOL
2115 LET A = 0
2120 RETURN
```

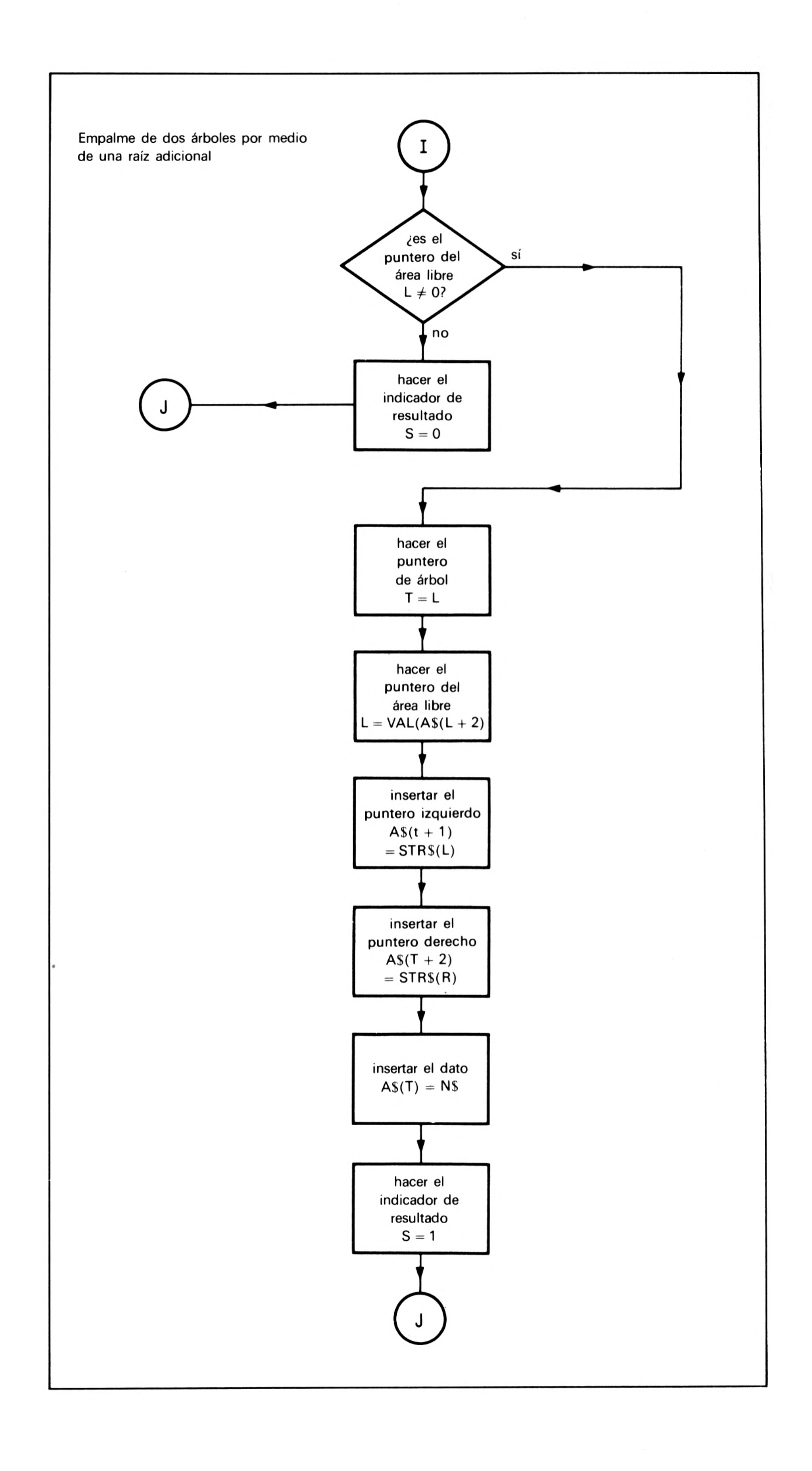


Entregar un indicador al subárbol izquierdo de un árbol dado

```
2200 REM ENTREGAR UN PUNTERO AL SUBARBOL IZ
         QUIERDO DE UN ARBOL DADO
2205 REM PARAMETROS
2210 REM A: PUNTERO DEL ARBOL
2215 REM L: PUNTERO DEL SUBARBOL IZQUIERDO
2220 REM S: INDICADOR DE ACIERTO FALLO
2225 IF A <> 0 THEN 2240
2230 LET S = 0
2235 RETURN
2240 LET L = VAL(A$(A + 1))
2245 LET S = 1
2250 RETURN
```

Entregar el valor del dato situado en la raíz

```
2300 REM ENTREGAR EL VALOR DEL DATO SITUADO EN LA
         RAIZ
2305 REM PARAMETROS
```


Empalme de dos árboles por medio de una raíz adicional
I
¿es el puntero del área libre L ≠ 0?
sí
no
hacer el indicador de resultado S = 0
J
hacer el puntero de árbol T = L
hacer el puntero del área libre L = VAL(A$(L + 2)
insertar el puntero izquierdo A$(t + 1) = STR$(L)
insertar el puntero derecho A$(T + 2) = STR$(R)
insertar el dato A$(T) = N$
hacer el indicador de resultado S = 1
J

```
2310 REM A: PUNTERO DE ARBOL
2315 REM N$: DATO
2320 REM S: INDICADOR DE ACIERTO/FALLO
2325 IF A <> 0 THEN 2340
2330 LET S = 0
2335 RETURN
2340 LET N$ = A$(A)
2345 LET S = 1
2350 RETURN
```

Empalmar dos árboles por medio de una raíz adicional

```
2400 REM EMPALMA DOS ARBOLES POR MEDIO DE UNA
         RAIZ ADICIONAL
2405 REM PARAMETROS
2410 REM I, D: PUNTEROS DE SUBARBOLES A LA
         IZQUIERDA Y DERECHA, RESPECTIVAMENTE
2415 REM A: PUNTERO DE ARBOL
2420 REM N$: DATO DEL NUDO DE LA RAIZ
2425 REM S: INDICADOR DE ACIERTO/FALLO
2430 IF L <> 0 THEN 2445
2435 LET S = 0
2440 RETURN
2445 LET A = L
2450 LET L = VAL(A$(L + 2))
2455 LET A$(A + 1) = STR$(I)
2460 LET A$(A + 2) = STR$(D)
2465 LET A$(A) = N$
2470 LET S = 1
2475 RETURN
```

Puntos de interés

- La rutina encargada de entregar un árbol vacío puede parecer trivial, pero es imprescindible para estructurar las operaciones siguientes en forma de módulos independientes y mantener así la estructura general del programa.

21.4 Algunas aplicaciones de los árboles

Los árboles se emplean en gran diversidad de situaciones de programación, de las que expondremos algunas de las más importantes. Algunos **sistemas de base de datos** mantienen ésta en una estructura jerárquica; aunque se trata de programas muy complejos, la estructura de datos básica es el árbol.

La aplicación más frecuente está relacionada con los **programas de compilación de lenguaje.** La sintaxis de los programas, las instrucciones y las expresiones aritméticas suelen analizarse dentro de una estructura de árbol antes de generar un código en el lenguaje al que se está traduciendo.

Algunos **sistemas operativos** dividen la memoria central del ordenador en una estructura de árbol en la que cada nudo contiene un segmento de código o un dato.

Merece la pena tener en cuenta que los módulos de muchos programas tienen una estructura parecida a la de árbol: en la raíz se sitúa el módulo de control, bajo ella los de operaciones y en los nudos del nivel inferior los de servicio.

21.5 Conclusión

En este capítulo hemos discutido la naturaleza y las propiedades de los árboles y hemos creado una serie de módulos que los manipulan. Dichos módulos constituyen además ejemplos de técnicas de diseño y estructuración de programas. En el ejercicio que sigue se especificarán más módulos de manipulación de árboles y programas que hacen uso de los mismos. Los puntos más importantes son:

- Un árbol es una estructura de datos jerarquizada en la que cada elemento o nudo está unido a otro situado a un nivel más alto, y a cero o más dispuestos por debajo de él.
- La porción de un árbol situada por debajo de un nudo se llama subárbol. Una de las propiedades más importantes de los árboles es que todos los subárboles tienen a su vez estructura de árbol.
- El árbol binario, en el que cada nudo tiene como máximo dos subárboles, es la forma de árbol más frecuente.
- Para realizar un árbol binario en Basic hacen falta tres elementos de una matriz por cada nudo: uno para el dato y dos para los indicadores correspondientes a los subárboles izquierdo y derecho.

Aquí termina la parte del libro dedicada a la programación de estructuras de datos fundamentales: pilas, colas, listas y árboles. Junto con las matrices, que se habían estudiado antes, y los registros, con los que trataremos en el próximo capítulo, constituyen los bloques constructivos de casi todas las estructuras de datos usadas en informática.

Un aspecto general que se desprende del estudio de todos estos

capítulos es que hay una diferencia entre las propiedades teóricas de una estructura de datos y las características de cualquier plasmación práctica de tales propiedades. Las limitaciones prácticas con que hemos topado en estos capítulos se deben en parte a la naturaleza del Basic y en parte a la capacidad finita de las memorias de los ordenadores.

El resto del libro irá dedicado a aplicaciones de la programación, y en algunas de ellas se utilizarán módulos creados en este capítulo y en los tres anteriores.

21
Ejercicio

1. Defina brevemente los siguientes términos: árbol, nudo, subárbol, raíz, árbol binario, recorrido de un árbol.
2. Numere los elementos de la matriz de la figura 21.4 a partir de 1 y determine los valores de los punteros.
3. Dibuje un esquema de un árbol equivalente a la lista del área libre de la figura 21.5.
4. Someta a un pase de prueba el módulo que empalma dos árboles por medio de un nudo adicional usando para ello los datos de la figura 21.6.
5. Añada al programa ejemplo 21.1 un módulo que entregue un puntero al subárbol derecho de un árbol dado.
6. Añada al programa ejemplo 21.1 un módulo que haga una lista de los elementos de la matriz encargada de almacenar los árboles. No es la forma más elegante de mostrar la estructura del árbol, pero basta a los efectos de este curso.
7. Vuelva a escribir los módulos del programa ejemplo 21.1 haciendo uso de la construcción **IF ... THEN ... ELSE** y agrupando varias instrucciones en una misma línea.
8. Se llama **recurrente** a un subprograma capaz de llamarse a sí mismo. Si tiene la suerte de contar con un ordenador que acepte el uso de subprogramas recurrentes en Basic, el algoritmo expuesto a continuación proporciona un método particularmente sencillo de recorrer un árbol.

   Recorre el árbol de izquierda a derecha, es decir, en el orden: subárbol izquierdo, nudo, subárbol derecho. Sirve para presentar en la salida y en el orden mencionado los elementos de un árbol. El algoritmo en cuestión es como sigue:

   recorrer el árbol;

   si el árbol no es nulo:

   recorrer el subárbol izquierdo;
   llevar a la salida el dato del nudo,
   recorrer el subárbol derecho.

   Pueden usarse los módulos del programa ejemplo 21.1 para obtener los punteros correspondientes a los subárboles izquierdo y derecho y el dato del nudo. El propio módulo necesita como parámetro un indicador orientado hacia el árbol que ha de recorrer.

Aun cuando no se disponga de un ordenador con posibilidad de utilizar subprogramas recursivos, merece la pena escribir un subprograma para este módulo y someterlo a un pase de prueba. Los datos apropiados pueden extraerse de las figuras 21.4 y 21.6.

9. En informática se usa con mucha frecuencia una notación para expresiones aritméticas y algebraicas llamada **notación polaca inversa** (que también se ha descrito en la pregunta 4 del ejercicio 18). A continuación damos algunos ejemplos de expresiones algebraicas escritas en las notaciones normal y polaca inversa:

Notación normal	*Notación polaca inversa*
$a + b$	$ab +$
$a \times (b - c)$	$abc - \times$
$(e - f) \div g + (h + j) \times k$	$ef - g \div hj + k \times +$

Observe que la notación polaca inversa hace innecesarios los paréntesis.

El objetivo del programa que se expondrá a continuación es introducir una expresión algebraica en notación polaca inversa y almacenarla en una estructura de árbol. Por ejemplo: la expresión polaca inversa correspondiente al árbol de la figura 21.3 es $ab \times c +$. El programa se estructura en tres niveles, uno de control, uno de operación y otro con varios módulos de servicio.

El nivel de control se encarga de introducir una cadena alfanumérica que representa una expresión en notación polaca inversa. Explora la cadena carácter por carácter y transfiere el control a un módulo de operación por cada carácter identificado. A continuación lleva a la salida la matriz que contiene el árbol o un recorrido de éste. El programa utiliza varios árboles y una pila y aprovecha los módulos del programa ejemplo 18.1.

Los módulos de operación son como sigue:

- Si el carácter es una **variable**, se carga en un **nudo terminal**, que es un árbol con subárboles nulos. A la vez se introduce en una pila un indicador orientado hacia el árbol.
- Si el carácter corresponde a una **operación** aritmética, se convierte en una raíz que empalma dos subárboles con indicadores extraídos de la pila: el situado en la cima se convierte en el indicador del subárbol derecho, y el situado bajo él pasa a ser el del izquierdo.

El nivel de servicio contiene los módulos de manipulación de pilas del programa ejemplo 18.1 y los de manipulación de árboles del programa ejemplo 21.1. Si se utiliza un módulo de recorrido del árbol para mostrarlo, la expresión aparece en notación algebraica normal, sin paréntesis.

10. Redacte la documentación para el programador y la guía del usuario correspondientes al programa escrito como respuesta a la pregunta 9.


RESET
RETURN

22
Proceso de datos comercial

Una de las aplicaciones más comunes de los ordenadores es el **proceso de datos comercial.** Aunque los ordenadores electrónicos se diseñaron en un principio para realizar cálculos científicos, sus posibilidades comerciales se hicieron evidentes muy pronto. Ya en 1950 había instalaciones de este género en funcionamiento y su uso se ha extendido hasta el extremo de que actualmente los sistemas comerciales informatizados son la norma y los manuales la excepción.

En esas aplicaciones se usan sobre todo lenguajes especializados, como el **Cobol** y el **RPG**, pero el Basic ha llegado a adquirir también una importancia considerable.

En este capítulo esbozaremos brevemente la naturaleza y el alcance del proceso de datos comercial y expondremos las áreas accesibles a programas escritos en Basic. El grueso del texto va dedicado al diseño y la escritura de un programa ejemplo que ilustra algunas de las características de esta clase de aplicaciones.

Se trata además del primer capítulo de una serie centrada en las aplicaciones de la programación. Depende en mayor o menor medida de todos los anteriores y, en particular, utiliza generosamente el material del número 10, que estudia la manipulación de ficheros. Sin embargo, lo estudiado aquí no volverá a desarrollarse en lo que queda de libro.

22.1

Naturaleza y alcance del proceso de datos comercial

El mundo del comercio abarca una gama muy amplia de actividades relacionadas todas ellas de forma directa o indirecta con los negocios y su financiación. El comercio exige el almacenamiento y el proceso de grandes cantidades de información. Antes de la llegada de los ordenadores, esa información se archivaba y trataba manualmente o por medio de sistemas electromecánicos.

El proceso de datos comercial se divide en cuatro áreas básicas: **contabilidad, control de existencias, información para la dirección** y **preparación de la nómina,** aunque en la práctica suelen difuminarse los límites entre unas y otras. En cualquier caso, todas estas actividades comparten una serie de características comunes, como son la conformidad a **métodos normalizados** y a exigencias legales, la necesidad de una **precisión** extrema y la obligación de atenerse a **fechas estrictas.**

Los métodos comerciales han seguido una larga evolución y gobiernan casi todos los aspectos del mundo mercantil, desde la terminología hasta el formato de los documentos. El proceso de datos se ve forzado a aceptar esas normas y esos métodos, aunque también ha influido sobre ellos en cierta medida. En este capítulo definiremos algunos términos y normas mercantiles.

La precisión es de importancia capital en cualquier sistema. Las fechas, cantidades y números de referencia deben almacenarse con una exactitud completa, porque la inexactitud en la contabilidad puede desencadenar la persecución legal de la firma afectada.

Toda actividad mercantil está gobernada por una serie de fechas: fechas de entrega de productos, fechas de pago, fechas de preparación de la nómina, etc. Como el ritmo de los negocios se ha acelerado, las fechas son cada vez más justas y los retrasos cada vez más caros.

Los sistemas de proceso de datos comerciales deben diseñarse con mucha atención para satisfacer todas las exigencias que acaban de esbozarse. El proceso de determinación de las exigencias concretas de una operación de proceso de datos y de las fases implicadas en la misma se llama **análisis de sistemas.** Los programas encargados de realizar esas fases, o algunas de ellas, no se escriben hasta que se ha completado ese análisis del sistema. Cada vez es más frecuente que, una vez terminado el proceso de análisis, se elijan una serie de programas previamente reunidos en una biblioteca.

En este capítulo daremos las especificaciones de los programas suponiendo que ya se ha procedido al análisis del sistema, porque lo que aquí nos interesa es asegurar que el diseño de tales programas satisface las especificaciones. Conviene tener presente que en la práctica, la programación forma parte de una actividad más amplia,

como es el diseño de sistemas comerciales de proceso de datos capaces de cumplir una serie de exigencias muy rigurosas.

22.2 Contabilidad

Definiremos en esta sección unos pocos de los términos usados habitualmente en **contabilidad,** con el fin de disponer de una información básica para la siguiente, en la que se describirá un programa encargado de realizar ciertas operaciones contables.

Cuando una empresa suministra bienes o servicios a un cliente, exige a éste el pago de los mismos por medio de un documento llamado **factura.** El proceso de imprimir o escribir tal documento se llama **extender una factura.**

Los términos del pago dependen de la clase de transacción. Cuando no se entrega el dinero inmediatamente, las cantidades de todas las facturas adeudadas se acumulan en una **cuenta** del cliente. Cada cierto tiempo, por lo general mensual o quincenalmente, el proveedor envía al cliente un **estado de cuentas** que recoge las cantidades devengadas por las facturas, los pagos que pudieran haberse efectuado y el **saldo**, que es la cantidad realmente adeudada. Las facturas se anotan en el **debe** de la cuenta y los pagos en el **haber**. Para certificar que se ha efectuado un pago se extiende un **recibo.**

Las cuentas de clientes no son las únicas que existen, pero todas tienen apartados reservados al debe (cantidades extraídas de la cuenta), al haber (cantidades sumadas a la cuenta) y al saldo, que es la diferencia entre todos los asientos de debe y haber a lo largo de cierto período. Cuando concluye un período, el saldo del mismo se lleva al siguiente en forma de **suma y sigue.** La figura 22.1 representa una factura, y la 22.2 una cuenta.

22.3 Programa ejemplo 22.1

El objetivo del programa es crear un **sistema de facturación** para una firma de venta por correo. El programa lleva a cabo las siguientes actividades:

Extender Este módulo del programa es el que extiende las facturas. Acepta y valida los datos de entrada para

	cuenta número 42163		factura	
	A.P. Blasco c/. Cerrajeros, 12 Almuñécar Granada		23751 21/02/81	
n.º artículo	descripción	cantidad	precio unitario	importe
37149	cable triple, 100 m	5	2.480	12.400
42159	clavija	20	640	12.800
42158	enchufe	20	560	11.200
			subtotal	36.400
	DJS Distribución Polígono industrial La Estación Torrejón de Ardoz MADRID		IVA (15%)	5.460
			total	41.860
			reg. IVA n.º 5726315	

Figura 22.1
Factura

	cuenta número 42163		estado de su cuenta	
	A.P. Blasco c/. Cerrajeros, 12 Almuñécar Granada		28/02/81	
fecha	referencia	debe	haber	saldo
01/02/81	suma y sigue			28.526
05/02/81	fact. 22462	10.616		39.142
09/02/81	recb. 14918		28.526	10.616
13/02/81	fact. 22597	21.800		32.416
21/02/81	fact. 23751	41.860		74.276
			Saldo deudor, ptas.	74.276
	DJS Distribución Polígono Industrial La Estación Torrejón de Ardoz MADRID			

Figura 22.2
Documento de estado de cuenta

cada factura y calcula las cantidades parciales y el total de las mismas.

Visualizar Visualiza la factura en curso.
Editar Permite corregir la factura en curso.
Archivar Archiva en un fichero la factura en curso.
Detener Cierra el fichero y pone fin al programa.

Los datos que deben introducirse para extender una factura son: **número de cuenta, nombre** y **dirección** del cliente, **fecha** y **número de líneas de artículos.** Cada línea de artículo consta de un **número de artículo**, su **descripción**, la **cantidad enviada** y el **precio unitario.** Tanto el número de cuenta como los números de los artículos tienen **cifras de verificación,** que examinaremos ahora mismo.

Nota sobre las cifras de verificación

Los números de cuenta y los números de artículo constan de cinco cifras, de las que la situada a la derecha sirve como **cifra de verificación.** El objetivo de esta cifra es garantizar que el número se ha tecleado correctamente, porque un error en los números de cuenta o de artículo podría acarrear consecuencias graves.

Las cifras de verificación se crean de diversas formas, y aquí describiremos una de las más frecuentes. Según este procedimiento, la cifra ha de ser tal que la suma de los productos de las cifras del número por una serie de **coeficientes de ponderación** sea múltiplo de 11. Si es preciso utilizar para ello el número 10, se representa mediante la letra X.

Los coeficientes de ponderación son 9, 7, 5, 3 y 1. Un ejemplo de verificación de un número de cuenta sería:

Número de cuenta:	1	2	4	8	X
Coeficientes de ponderación:	9	7	5	3	1
Productos:	9 + 14 + 20 + 24 + 10 = 77				

La suma de los productos es múltiplo de 11. Las cifras de verificación calculadas de esta forma se llaman cifras de verificación **ponderadas módulo 11.**

Diseño del programa

El diseño se realiza siguiendo el proceso de refinamiento progresivo. En una primera fase se especifica la estructura general del programa en términos de sus módulos y se esboza la función de cada uno de ellos.

Primer refinamiento

Es obvio que hace falta un **módulo de control** para interpretar las órdenes y transferir el mando a un sitio u otro en función de ellas. El siguiente nivel está ocupado por los **módulos de operaciones,** en número de uno por cada orden, y por un **módulo de inicialización**

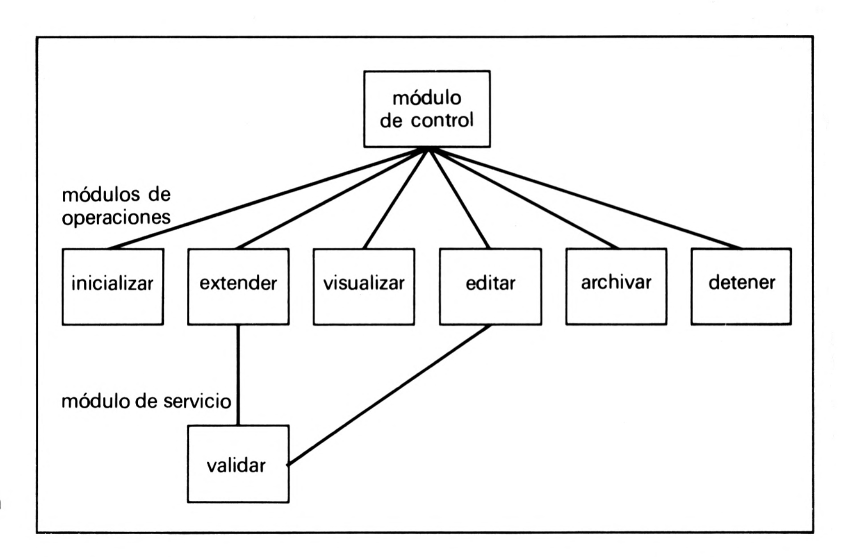


Figura 22.3
Estructura general del programa ejemplo 22.1

encargado de crear el fichero de facturas. En el nivel inferior se encuentra el **módulo de validación,** al que llaman las rutinas de extensión y edición de facturas. La figura 22.3 recoge la estructura general descrita.

Segundo refinamiento

En esta segunda etapa de refinamiento se deciden las estructuras de datos que van a usarse. Cuando los datos de cada factura se almacenan en el correspondiente fichero constituyen **registros.** Por desgracia, el Basic no soporta registros de datos en la memoria principal.

Una solución de compromiso es utilizar varias matrices, una por cada columna de la factura. Según esto, habrá una matriz de números de artículo, otra de descripciones, otra de cantidades, otra de precios unitarios y una última de importes (importe = precio unitario × cantidad). Una línea de artículo está formada por un elemento de cada una de esas matrices identificado en todas ellas por el mismo índice.

Las formaciones descritas y unos pocos datos individuales mantienen la factura en curso durante la entrada, la presentación, la edición y el archivado. Si se introduce a continuación una nueva factura, ésta se superpone a la anterior.

Tercer refinamiento

En la tercera y última etapa de refinamiento se especifica el funcionamiento de cada uno de los módulos, elaborando un algoritmo del mismo cuando sea necesario.

Inicializar

Se declaran las matrices encargadas de mantener la factura en curso y se crea el fichero en el que se archivarán las facturas. Se introduce la fecha y se lee la matriz de coeficientes de ponderación.

Extender

El final de la introducción de una factura viene indicado por un número de cuenta igual a cero. De acuerdo con esto, el módulo seguirá los siguientes pasos:

Introducir el nombre y la dirección del cliente y el número de cuenta.
Validar el número de cuenta y, en caso necesario, volver a introducirlo.
Introducir un número de artículo.
Mientras el número de artículo no sea cero y mientras las matrices no están llenas,
Repetir el proceso:

> Validar el número de artículo y, en caso necesario, volver a introducirlo.
> Introducir descripción, cantidad y precio unitario.
> Calcular y presentar el importe.
> Sumar el importe al total de la factura.
> Introducir un número de artículo.

Visualizar el total de la factura.

Validar

Se calcula la suma de los productos de las cifras por los coeficientes de ponderación. Si el resultado es múltiplo exacto de 11, el número es válido. En caso contrario, no lo es.

Visualizar

Se visualiza toda la información de la factura en curso, organizada de la forma adecuada.

Editar

Por medio de una serie de preguntas y mensajes que aparecen en pantalla, se invita al usuario a corregir cualquiera de los datos contenidos en la factura. Si la corrección afecta a los números de cuenta o de artículo, se llama al módulo de validación.

Archivar

Todos los datos de la factura se archivan en un registro del fichero de facturas. El registro incluye un campo para el número de líneas de artículos y un indicador de fin de registro.

Detener

Se escribe un registro de fin de fichero en el de facturas, que a continuación se cierra.

Variables

Globales:

F$	Fecha.
A$	Número de cuenta.
N$	Nombre del cliente.
R$	Dirección del cliente.
I$(20)	Matriz de números de artículo.
E$(20)	Matriz de descripciones.
C(20)	Matriz de cantidades.
P(20)	Matriz de precios.
M(20)	Matriz de importes.
T	Total de la factura.
W(5)	Matriz de coeficientes de ponderación.
K	Número de líneas de artículos de la factura.

Parámetros:

X$	Parámetro para el módulo de validación.
V	Indicador de validez: **V** = 1, válido; **V** = 0, no válido.

Locales:

B$	Introducir número de artículo.
I	Contador de bucle.
O$	Orden.
D	Suma de los productos del módulo de validación.

Diagrama de flujo

La figura 22.4 ilustra el flujo de control de algunos de los módulos del programa ejemplo 22.1.

Programa

Sólo se han escrito los módulos de inicialización, extensión, archivo, detención y validación. La redacción de los demás se deja como ejercicio para el lector.

```
1000 REM PROGRAMA EJEMPLO 22.1
1005 REM SISTEMA DE FACTURACION
1010 REM
```

Módulo de control

```
2000 REM MODULO DE CONTROL
2005 GOTO 3000: REM INICIALIZACION
2010 REM INICIO DEL BUCLE DE CONTROL
2015 PRINT "ORDEN?";
2020 INPUT O$
2025 IF O$ = "EXTENDER" THEN 4000
2030 IF O$ = "VISUALIZAR" THEN 5000
2035 IF O$ = "EDITAR" THEN 6000
2040 IF O$ = "ARCHIVAR" THEN 7000
2045 IF O$ = "DETENER" THEN 8000
2050 PRINT "ORDEN NO VALIDA. POR FAVOR VUELVA A
           INTRODUCIRLA"
2055 GOTO 2010
2060 REM
```

Módulo de inicialización

```
3000 REM MODULO DE INICIALIZACION
3005 DIM I$(20), E$(20), C(20), P(20), M(20),
        W(5)
3010 REM LEER FACTORES DE PESO
3015 FOR I = 1 TO 5
3020 READ W(I)
3025 NEXT I
3030 DATA 9, 7, 5, 3, 1
3035 REM CREACION DE UN FICHERO DE FACTURAS
3040 OPEN "FICHEROFACTURAS" FOR APPEND AS #10
3050 REM VISUALIZAR ENCABEZAMIENTO E INTRODUCCION
        DE FECHAS
3055 PRINT "SISTEMA DE FACTURACION"
3060 PRINT
```

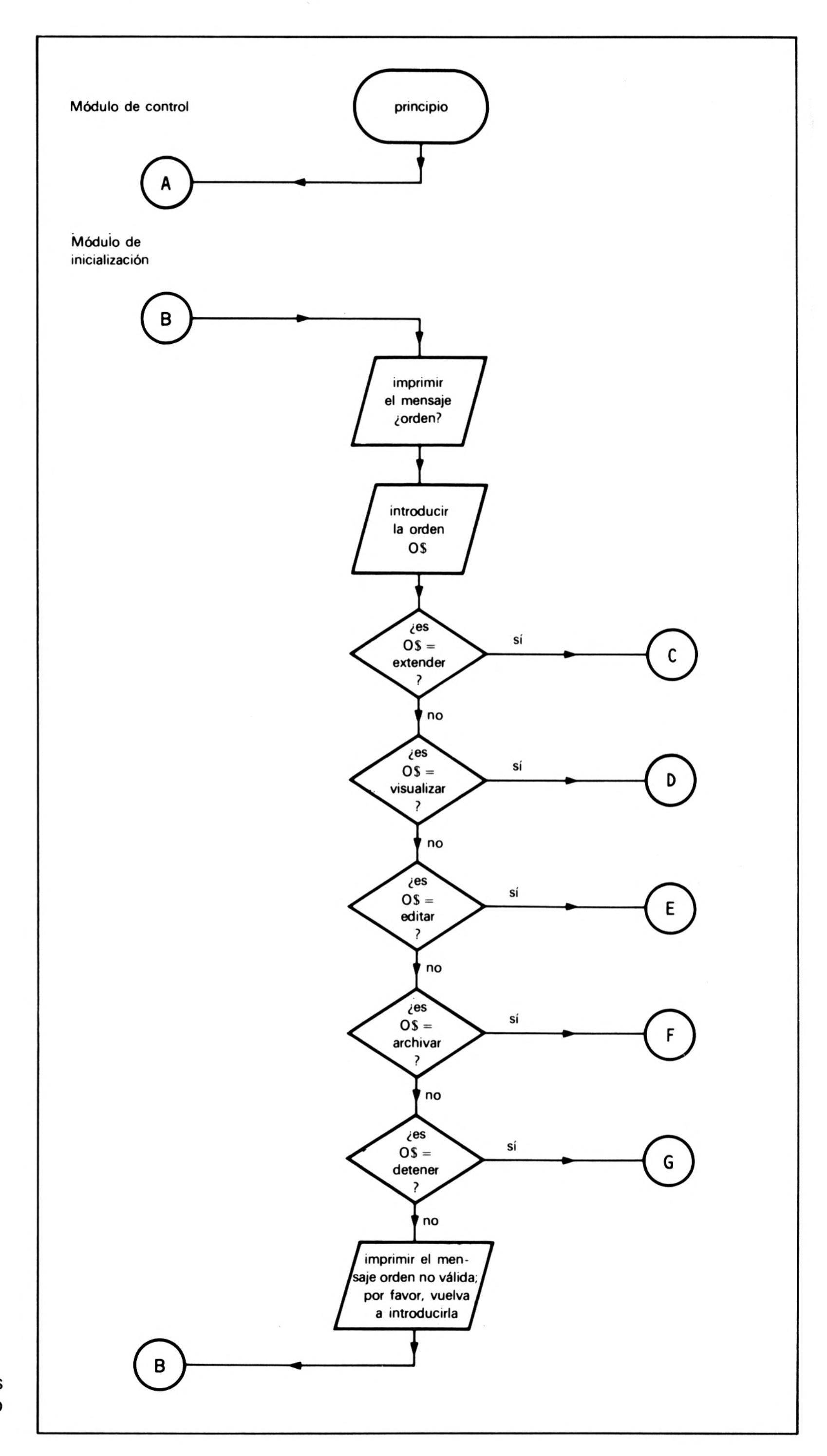
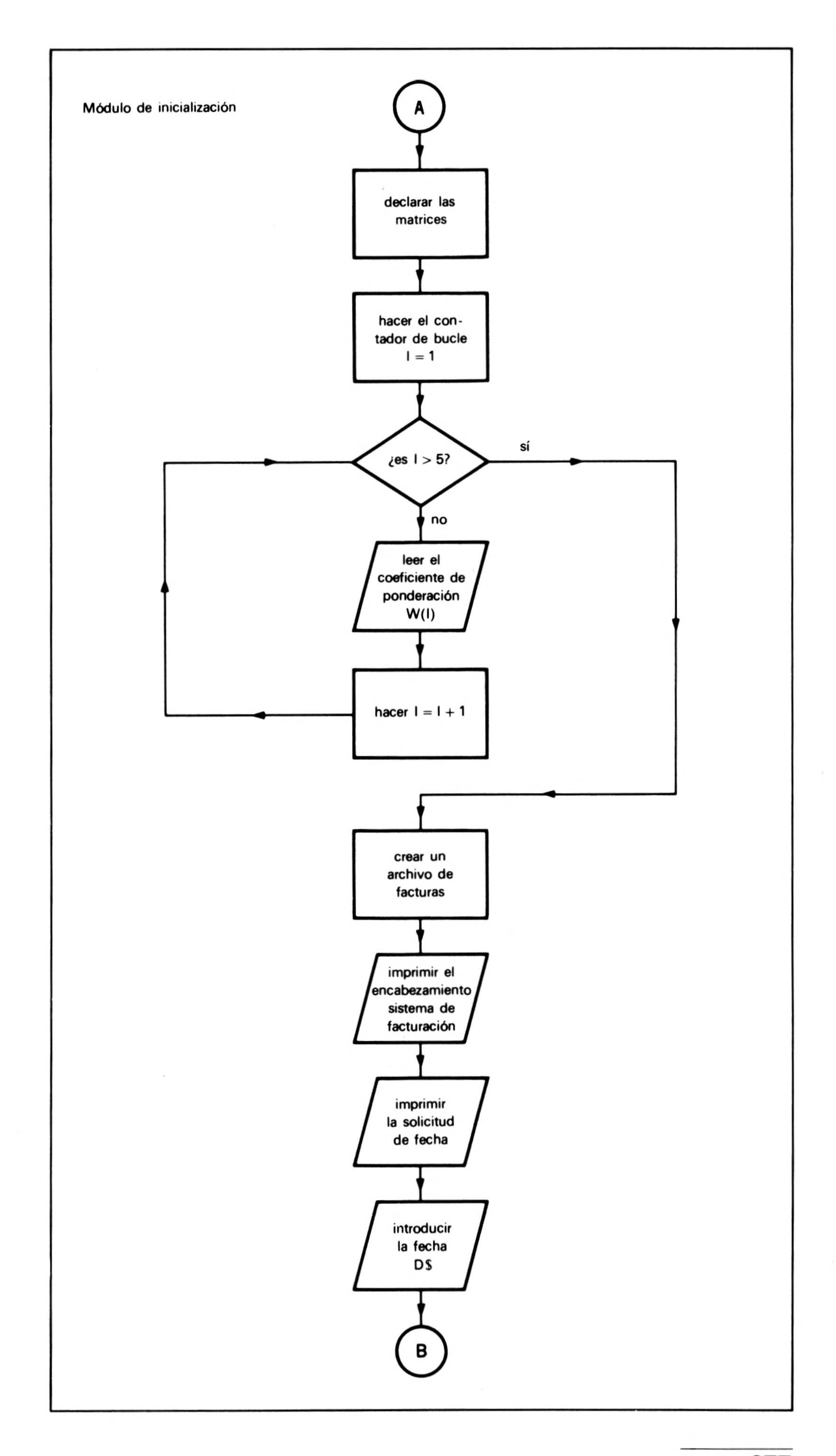
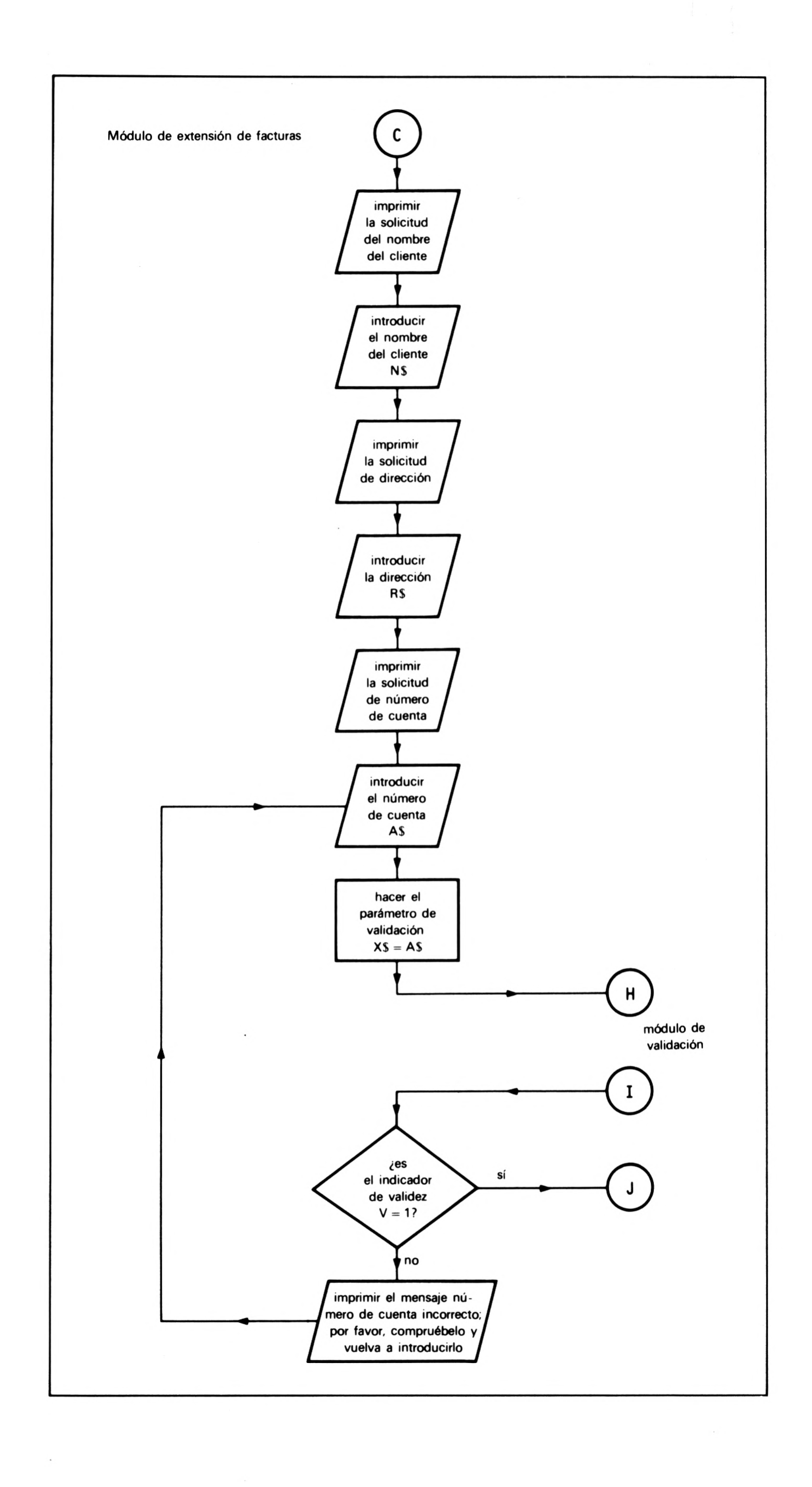
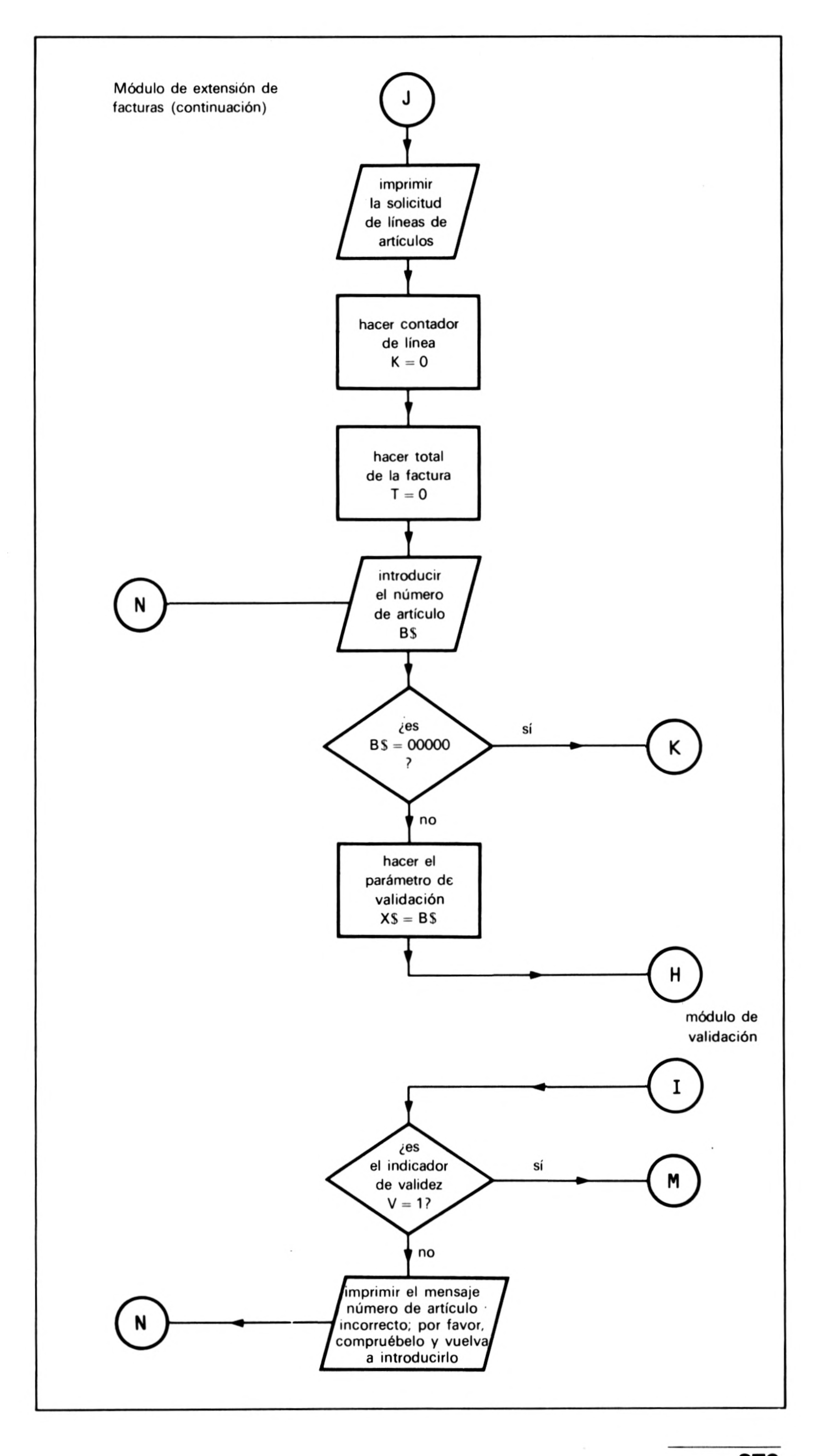


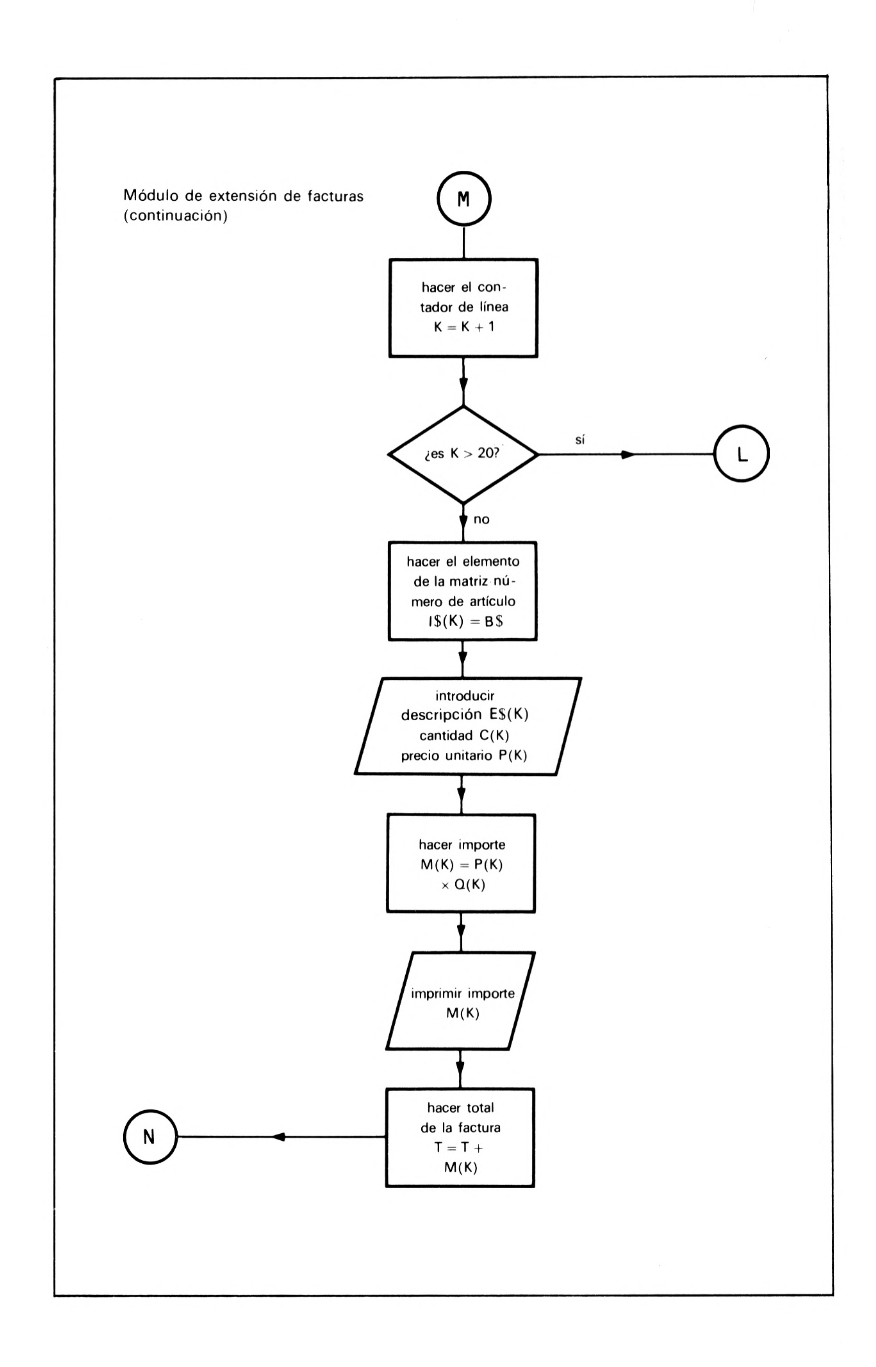
Figura 22.4
Diagrama de flujo de los módulos elegidos del programa ejemplo 22.1


Módulo de inicialización
A
declarar las matrices
hacer el contador de bucle I = 1
¿es I > 5?
sí
no
leer el coeficiente de ponderación W(I)
hacer I = I + 1
crear un archivo de facturas
imprimir el encabezamiento sistema de facturación
imprimir la solicitud de fecha
introducir la fecha D$
B

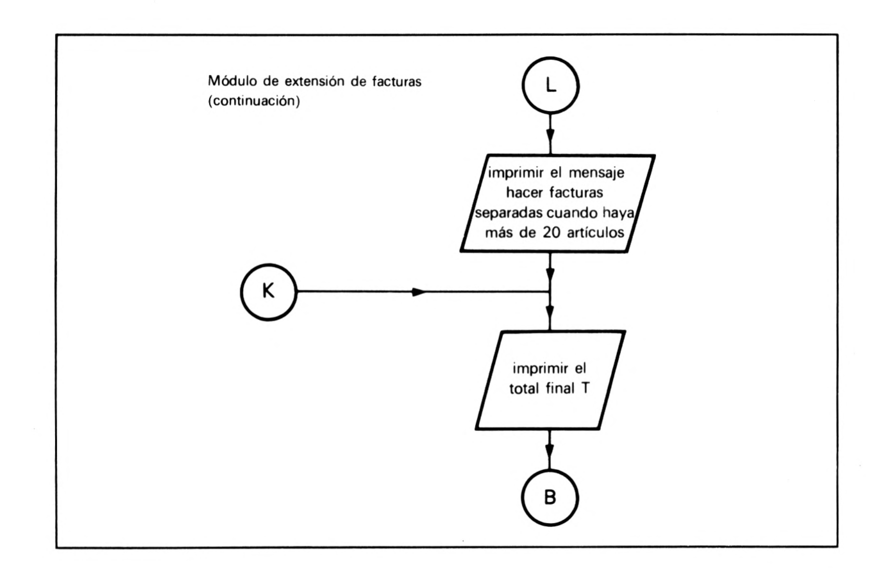

Módulo de extensión de facturas
C
imprimir
la solicitud
del nombre
del cliente
introducir
el nombre
del cliente
N$
imprimir
la solicitud
de dirección
introducir
la dirección
R$
imprimir
la solicitud
de número
de cuenta
introducir
el número
de cuenta
A$
hacer el
parámetro de
validación
X$ = A$
H
módulo de
validación
I
¿es
el indicador
de validez
V = 1?
sí
J
no
imprimir el mensaje nú-
mero de cuenta incorrecto;
por favor, compruébelo y
vuelva a introducirlo



Módulo de extensión de facturas (continuación)
J
imprimir la solicitud de líneas de artículos
hacer contador de línea K = 0
hacer total de la factura T = 0
N
introducir el número de artículo B$
¿es B$ = 00000 ?
sí
K
no
hacer el parámetro de validación X$ = B$
H
módulo de validación
I
¿es el indicador de validez V = 1?
sí
M
no
imprimir el mensaje número de artículo incorrecto; por favor, compruébelo y vuelva a introducirlo
N

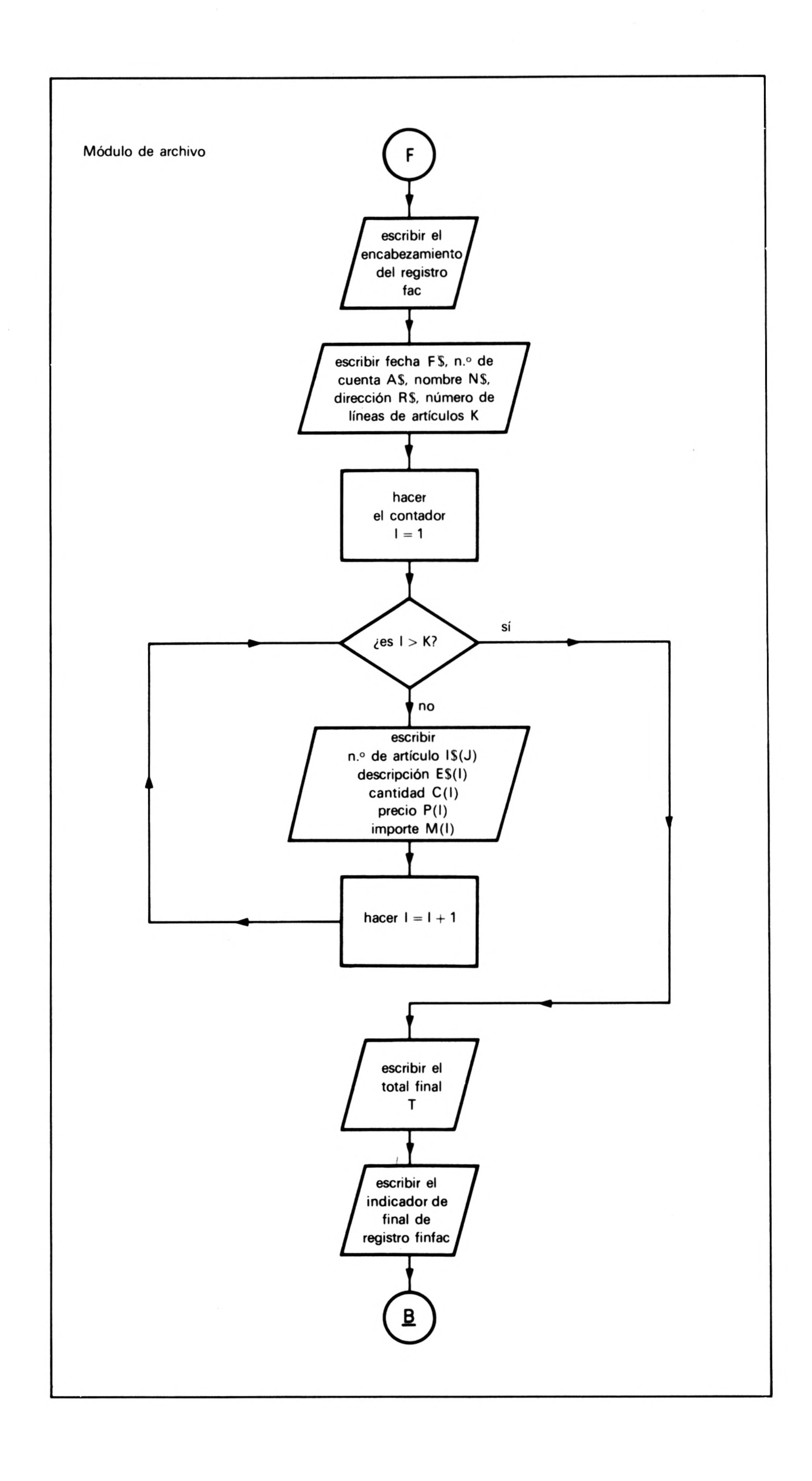



```
3065 PRINT "INTRODUZCA FECHA";
3070 INPUT F$
3075 PRINT
3080 GOTO 2010
3085 REM
```

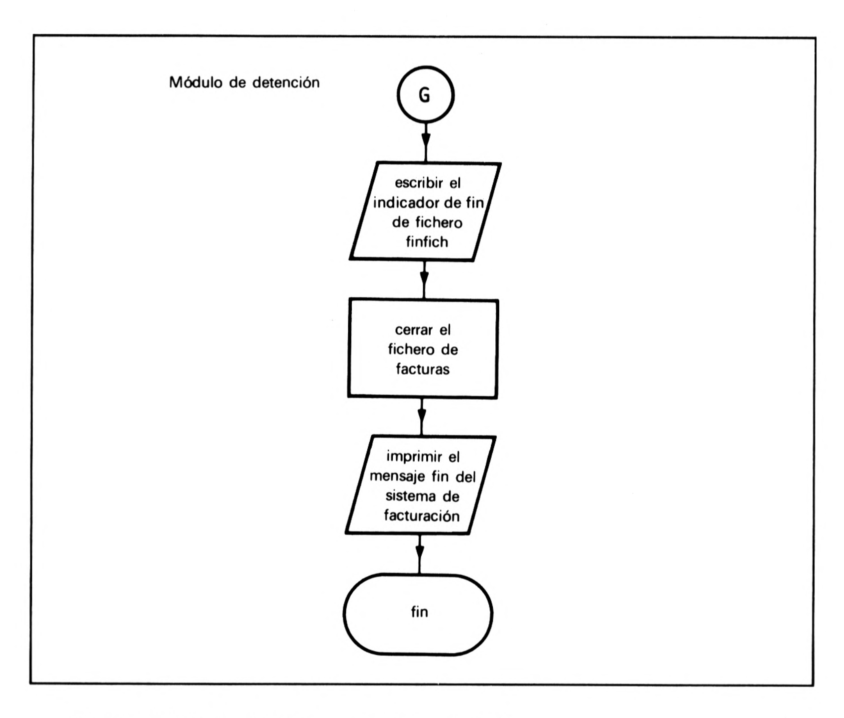


Módulo de extensión de facturas

```
4000 REM MODULO DE EXTENSION DE FACTURAS
4005 PRINT "NOMBRE DEL CLIENTE";
4010 INPUT N$
4015 PRINT "DIRECCION";
4020 INPUT R$
4025 PRINT "NUMERO DE CUENTA";
4030 INPUT A$
4035 LET X$ = A$
4040 GOSUB 9000: REM VALIDAR NUMERO DE CUENTA
4045 IF V = 1 THEN 4100
4050 PRINT "NUMERO INCORRECTO DE CUENTA"
4055 PRINT "POR FAVOR VERIFIQUE Y VUELVA A INTRO
          DUCIRLO"
4060 GOTO 4030
4065 REM
4100 PRINT "LINEAS DE ELEMENTOS"
4105 LET K = 0
4110 LET T = 0
4115 INPUT B$
4120 IF B$ = "00000" THEN 4300
4125 LET X$ = B$
4130 GOSUB 9000: REM VALIDA A NUMERO DE ARTICULO
4135 IF V = 1 THEN 4155
4140 PRINT "NUMERO DE ELEMENTO INCORRECTO"
4145 PRINT "POR FAVOR VERIFIQUE Y VUELVA A INTRO
          DUCIRLO"
4150 GOTO 4115
4155 LET K = K + 1
4160 IF K > 20 THEN 4200
4165 LET I$(K) = B$
```



Módulo de archivo
F
escribir el encabezamiento del registro fac
escribir fecha F$, n.º de cuenta A$, nombre N$, dirección R$, número de líneas de artículos K
hacer el contador I = 1
¿es I > K?
sí
no
escribir n.º de artículo I$(J) descripción E$(I) cantidad C(I) precio P(I) importe M(I)
hacer I = I + 1
escribir el total final T
escribir el indicador de final de registro finfac
B



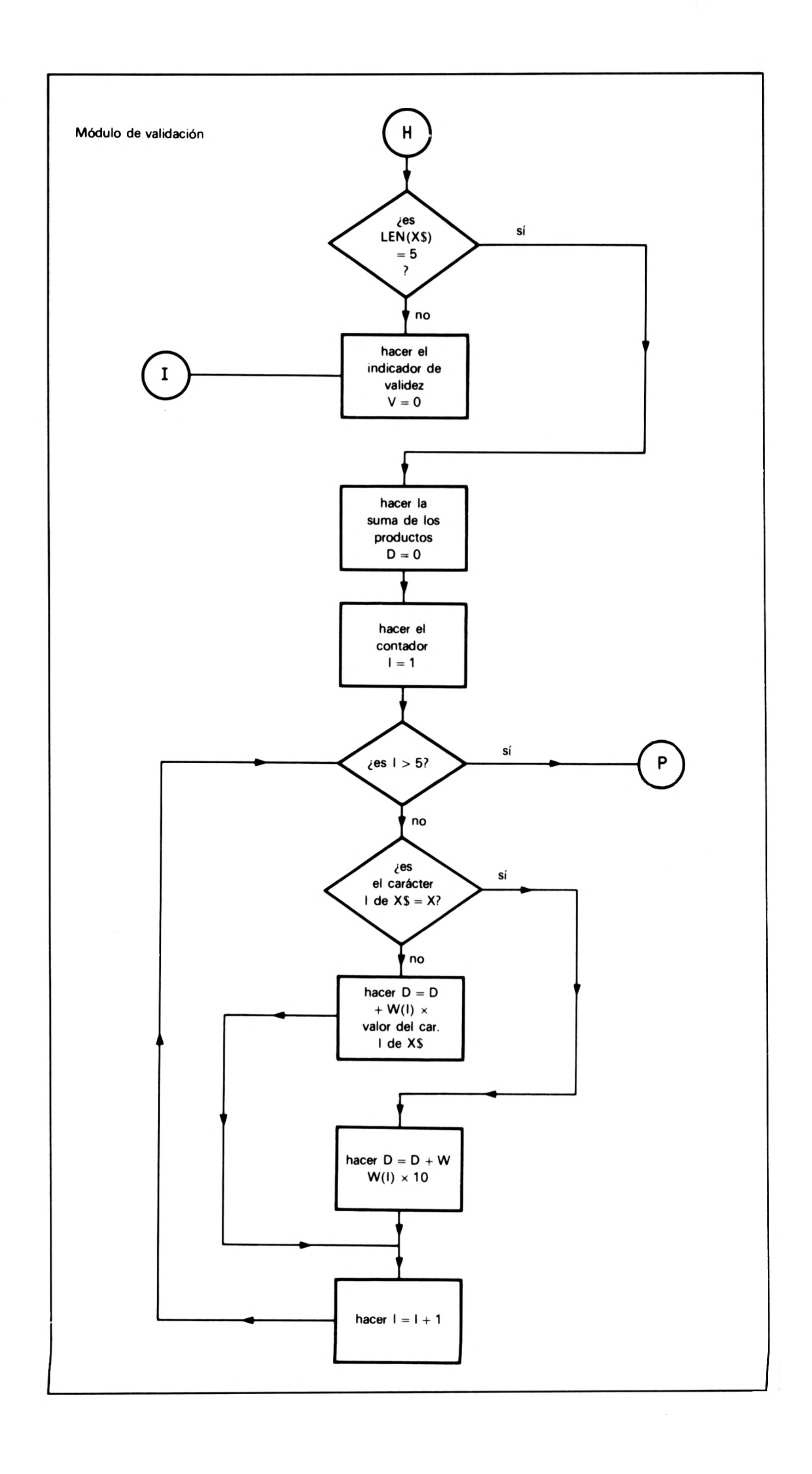

```
4170 INPUT E$(K), C(K), P(K)
4175 LET M(K) = C(K) * P(K)
4180 PRINT "CANTIDAD"; M(K)
4185 LET T = T + M(K)
4190 GOTO 4115
4195 REM
4200 PRINT "UTILICE FACTURAS SEPARADAS PARA"
4205 PRINT "MAS DE 20 ELEMENTOS"
4210 PRINT
4215 REM
4300 PRINT "TOTAL DE FACTURAS";T
4305 PRINT
4310 GOTO 2010
4315 REM
```

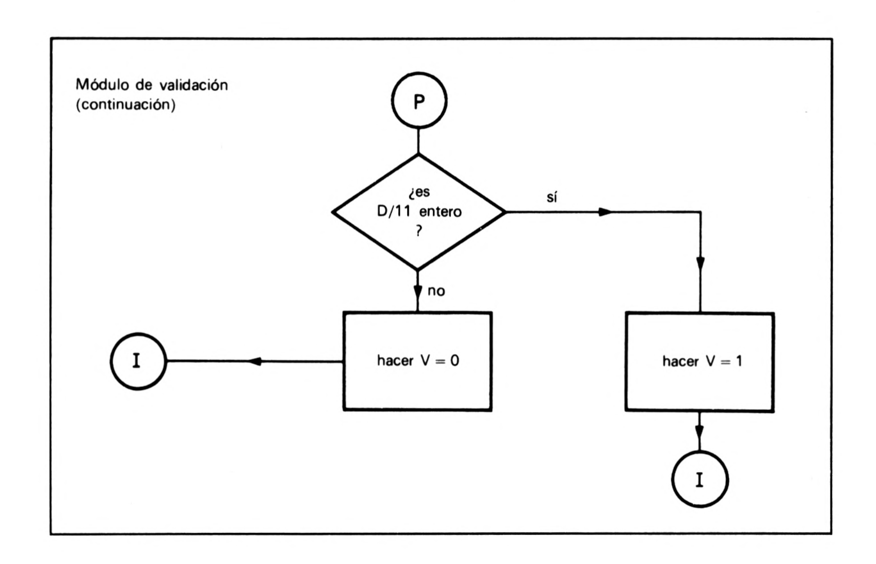
Módulo de visualización

```
5000 REM MODULO DE VISUALIZACION
5005 REM DEBE SER ESCRITO COMO UN EJERCICIO
5010 GOTO 2010
5015 REM
```

Módulo de edición

```
6000 REM MODULO DE EDICION
6005 REM DEBE SER ESCRITO COMO UN EJERCICIO
6010 GOTO 2010
6015 REM
```


Módulo de validación
H
¿es
LEN(X$)
= 5
?
sí
no
hacer el
indicador de
validez
V = 0
I
hacer la
suma de los
productos
D = 0
hacer el
contador
I = 1
¿es I > 5?
sí
P
no
¿es
el carácter
I de X$ = X?
sí
no
hacer D = D
+ W(I) ×
valor del car.
I de X$
hacer D = D + W
W(I) × 10
hacer I = I + 1



Módulo de archivado

```
7000 REM ESCRIBIR LAS FACTURAS EN EL ARCHIVO
7005 WRITE #10, "FACT": REM ENCABEZAMIENTO DEL
          REGISTRO
7010 WRITE #10, F$, A$, N$, R$, K
7015 FOR I = 1 TO K
7020 WRITE #10, I$(I), E$(I), C(I), P(I), M(I)
7025 NEXT I
7030 WRITE #10, T
7035 WRITE #10, "FINFACT": REM INDICADOR DE FINAL
          DE REGISTRO
7040 GOTO 2010
7045 REM
```

Módulo de detención

```
8000 REM MODULO DE DETENCION
8005 WRITE #10, "FINARCHIVO": REM INDICADOR
          DE FINAL DE REGISTRO
8010 CLOSE #10
8020 PRINT "FINAL DEL SISTEMA DE FACTURAS"
8025 PRINT
8030 STOP
```

Módulo de validación

```
9000 REM MODULO DE VALIDACION
9005 REM PARAMETROS
9010 REM X$: NUMERO DE CUENTA O ELEMENTO
9015 REM V: INDICADOR DE VALIDEZ O NO
```

```
9020 IF LEN(X$) = 5 THEN 9035
9025 LET V = 0
9030 RETURN
9035 LET D = 0
9040 FOR I = 1 TO 5
9045 IF MID$(X$, I, 1)= "X" THEN 9060
9050 LET D = D + W(I) * VAL(MID$(X$, I, 1))
9055 GOTO 9065
9060 LET D = D + W(I) * 10
9065 NEXT I
9070 IF INT(D/11) = D/11 THEN 9085
9075 LET V = 0
9080 RETURN
9085 LET V = 1
9090 RETURN
9095 REM
```

Puntos de interés

- Los módulos de operaciones no se han escrito como subrutinas, porque no devuelven el control al punto desde el que fueron llamados, sino a otro.
- El bucle principal del módulo encargado de extender las facturas constituye una interpretación bastante libre del algoritmo "repetir mientras".
- Varios módulos del programa contienen instrucciones de manipulación del fichero que dependen del ordenador en uso.

22.4 Conclusión

En este capítulo hemos estudiado algunos términos y conceptos relacionados con el proceso de datos comercial y hemos visto la aplicación de algunos de ellos en un programa escrito en Basic. Los puntos más importantes son:

- El proceso de datos comerciales mediante ordenador es la forma más nueva de realizar unas tareas que ya tienen varios siglos de antigüedad.
- Los programas comerciales deben atenerse a las normas y usos mercantiles.
- El diseño de un sistema de proceso de datos comercial es el resultado de un proceso conocido como análisis de sistemas. La redacción de los programas es la etapa final del proceso de diseño.

- Los sistemas comerciales deben ser rigurosamente exactos y estar en condiciones de satisfacer las fechas de vencimiento de las operaciones.

22

Ejercicio

1. Defina brevemente los siguientes términos: análisis de sistemas, factura, cuenta, estado de cuenta, saldo, debe, haber, recibo.
2. Escriba los módulos que faltan al programa ejemplo 22.1. Antes de hacerlo amplíe sus especificaciones y, si es preciso, utilice módulos de servicio adicionales.
3. Redacte la documentación para el programador y la guía para el usuario correspondientes al programa ejemplo 22.1.
4. Cambie el módulo de validación del programa ejemplo 22.1 por otro que trabaje con arreglo a los siguientes principios:

    El número de factura o de cuenta está formado por cinco cifras seguidas de una letra mayúscula que sirve como elemento de verificación. Para determinar la letra se obtiene la suma ponderada de los productos de las cifras y el resultado se divide por 26; si el resto de la división es cero, la letra de verificación es la A, si es 1, la B, etc. Se utilizan los mismos coeficientes de ponderación del módulo original.

5. Muchas de las modernas cadenas comerciales han sustituido las cajas registradoras tradicionales por terminales de **punto de venta.** Los datos que se introducen en dichos terminales cada vez que se efectúa una venta permiten actualizar en tiempo real los ingresos de efectivo y los niveles de existencias.

    El funcionamiento es el siguiente: todos los artículos están identificados por un número de almacén, que debe ser un entero positivo. En la memoria central del ordenador se mantiene una lista con los precios y existencias de cada uno de los artículos. La realización de una venta se registra como sigue:

    Por cada artículo o grupo de artículos identificados por un mismo número

        se introduce ese número,
        se accede al precio y se presenta en pantalla,
        se introduce la cantidad,
        se actualiza el nivel de existencias de ese artículo,
        se calcula el total parcial, se presenta en pantalla y se suma al importe total de la compra,

    se presenta en pantalla el total de la compra,
    se introduce la cantidad de dinero entregada por el comprador,
    se calcula el cambio y se presenta en pantalla.

    Escriba un programa que simule el funcionamiento de un terminal de punto de venta como el que acaba de describirse.

    El programa puede ampliarse para hacer que lea los precios y niveles de existencias contenidos en una memoria auxiliar antes de empezar la

operación y para que transfiera los valores actualizados de dichos datos a otro fichero al cerrarla.

6. Modifique el programa ejemplo 22.1 para que genere un **número de factura** por cada factura. Los números han de ir en orden ascendente, empezando por uno que se introduce junto con la fecha. Deben también incluir una cifra de verificación calculada como ya se ha explicado a propósito de los números de cuenta y artículo. El número de factura se escribe en el fichero con los demás datos de aquélla.

7. Modifique el programa ejemplo 22.1 para que calcule el IVA correspondiente al total de la factura y lo sume al mismo. Al iniciar el programa, se introduce el porcentaje del IVA.

8. En la práctica, el sistema de facturación del programa ejemplo 22.1 no sería sino una parte de un paquete completo de contabilidad. Especificaremos en esta pregunta otra parte del mismo paquete: la encargada de preparar los estados de cuentas a partir del fichero creado por el sistema de facturación.

El programa responde a las siguientes órdenes:

Preparar	Se introduce el número de la cuenta cuyo estado se desea conocer. Se busca secuencialmente en el fichero de facturas y se crea una línea del estado de cuenta a partir de cada una de las facturas que lleven el número introducido al principio.
Visualizar	Se visualiza el estado de la cuenta.
Archivar	El estado de la cuenta en curso se escribe en un fichero de cuentas.
Detener	Se cierra el fichero de cuentas y se detiene el programa.

En el estado de la cuenta figuran el número de cuenta, el nombre y la dirección del cliente y la fecha, más una línea por cada factura. Cada línea contiene a su vez una fecha, un número de factura, el total de esa factura y el total parcial. La cantidad total adeudada aparece en la parte inferior del documento. El programa puede incluir también una rutina de cálculo de un descuento aplicado sobre el total final adeudado.

9. El paquete de contabilidad desarrollado en el programa ejemplo 22.1 y en la pregunta 8 de este ejercicio no tiene en cuenta los **recibos.** Esta pregunta especifica una ampliación del sistema pensada para manejarlos.

Para ello se amplía el programa de facturación de manera que sea capaz de extender y archivar recibos como respuesta a las siguientes órdenes:

Recibir	Se aceptan y validan los datos introducidos con destino a un recibo.
Archivar recibo	El recibo en curso se archiva en el fichero de facturas, que pasa así a convertirse en un fichero mixto de facturas/recibos.

Es imprescindible que el recibo tenga la misma estructura que una factura de una línea, porque en caso contrario el programa de contabilidad no podría leer el registro del mismo. Para extender un recibo hacen falta los siguientes datos: número de cuenta, nombre y dirección del cliente, fecha y una línea de artículo con una descripción (metálico, cheque, etc.) y una cantidad. Al archivar el recibo, los otros campos de la factura se rellenan con series nulas o ceros. Hay que elegir también cabeceras e indicadores de fin de registro adecuados. El programa de

contabilidad lee en el fichero las facturas y los recibos. Hay que corregir el estado de cuentas para incluir en el mismo columnas de debe y haber (véase la figura 22.2).

10. El paquete de contabilidad desarrollado en las preguntas 8 y 9 puede ampliarse todavía más para que realice algunas de las operaciones siguientes o todas ellas:

Cargar un fichero de facturas y recibos en la memoria principal y clasificarlo en orden de números de cuenta.

Localizar todas las facturas y recibos correspondientes a un número de cuenta determinado siguiendo una técnica de búsqueda binaria.

Recorrer el fichero ordenado y preparar los estados de todas las cuentas.

Transferir el fichero ordenado de facturas y recibos a una memoria auxiliar.

11. Cuando hay que insertar y extraer con mucha frecuencia registros de un fichero, lo mejor es cargarlo en una estructura resultante de la ampliación del concepto de lista expuesto en el capítulo 20. Cada registro del fichero contiene una serie de campos de datos y un indicador que señala hacia el siguiente registro. La inserción y extracción de registro se hace como ya se expuso en el mencionado capítulo 20.

a) Modifique los módulos del programa ejemplo 20.1 para que manipule un fichero estructurado en forma de lista encadenada, suponiendo que todo él está contenido en una matriz almacenada en la memoria principal.

b) Ampliar el juego de módulos para incluir rutinas capaces de transferir el fichero desde la memoria auxiliar hasta la matriz de la memoria principal, y viceversa.

c) Escribir un programa principal y, si es necesario, un nivel de módulos de operaciones para realizar una aplicación que haga uso de los módulos de manipulación de ficheros de las partes *a*) y *b*). Una aplicación adecuada sería el mantenimiento de la lista de personas alojadas en un gran hotel; a cada una de dichas personas correspondería un registro con campos para su nombre, número de habitación, domicilio, fecha de entrada y fecha de salida esperada. Los registros se añaden al fichero cuando llega el viajero y se eliminan del mismo cuando se marcha.


RESET
RETURN

23 Programas interactivos

El Basic es uno de los lenguajes más adecuados para escribir los llamados **programas interactivos,** que estudiaremos en este capítulo. Empezaremos por exponer algunas de sus características para a continuación resumir los elementos que hacen al Basic tan apropiado para este tipo de programas. La mayor parte del capítulo se dedica a un programa ejemplo.

Se utilizará aquí material de casi todos los capítulos precedentes, aunque este tema tiene poco que ver con el proceso de datos comercial que acabamos de estudiar en el último capítulo. En efecto, los programas interactivos no están limitados por los usos, normas e imposiciones legales que rigen el mundo del comercio, aunque los principios de diseño siguen siendo aplicables en este campo.

23.1 Características de los programas interactivos

Se llaman interactivos a los programas que permiten entablar una "conversación" entre el ordenador y la persona que lo utiliza. No es una definición muy rigurosa, pero resulta adecuada. Muchos de los

programas que hemos estudiado hasta ahora son, en cierta medida, interactivos, pero este capítulo se centrará en los que tienen esa característica en el mayor grado.

Los programas interactivos se caracterizan, por tanto, por una cantidad considerable de operaciones de entrada y salida, que con frecuencia incluyen gráficos. El proceso suele ser sencillo, y ha de ser rápido para no hacer esperar al usuario. En otras palabras: los programas interactivos son programas en **tiempo real.**

Cubren estos programas un abanico de actividades muy amplio: recuperación de datos, simulación de vuelo, programas de enseñanza mediante la práctica y juegos.

De particular importancia es el **interfaz con el usuario** del programa, constituido por la información que se entrega y que se solicita a aquél. Este interfaz ha de diseñarse con mucho cuidado, teniendo en cuenta las posibilidades y limitaciones de los supuestos usuarios. A este respecto, son aplicables las observaciones hechas en las secciones 13.6 a 13.8 a propósito de la documentación para el usuario. La misma importancia tienen las **instrucciones iniciales,** que explican la finalidad del programa y su funcionamiento, las **indicaciones,** que dicen al usuario lo que debe hacer a continuación, y los **mensajes de error** presentados en respuesta a una operación de entrada incorrecta por parte del usuario. Esta información ha de ser completa, concisa y clara.

23.2

Basic: un lenguaje interactivo

El Basic se creó desde un principio como lenguaje interactivo, tanto desde el punto de vista del desarrollo de los programas como desde la perspectiva del tipo de programas escritos. La mayor parte de los sistemas operativos que soportan el Basic permiten cargar, corregir y utilizar los programas por medio de un proceso de conversación. Igualmente, el Basic tiene varias características que lo hacen muy apropiado para escribir programas interactivos. A este respecto tienen especial importancia sus sencillos, pero potentes, recursos de entrada y salida; las posibilidades de generación de gráficos de muchas versiones modernas constituyen una ventaja adicional.

En este capítulo escribiremos un programa interactivo en la versión de referencia del Basic, y en el próximo discutiremos los recursos gráficos de algunas versiones.

23.3

Programa ejemplo 23.1

El objetivo de este programa es crear un paquete de aritmética práctica sencillo adecuado para niños. El paquete permite practicar las cuatro reglas con enteros positivos comprendidos entre 1 y 100. Se presenta una secuencia de cuentas con sólo dos números y se invita al usuario a dar el resultado de cada una. Si la respuesta es incorrecta, se presenta en pantalla el resultado correcto. Hay además un marcador que acumula el número de respuestas correctas. Tras cada operación, se pregunta al usuario si quiere hacer otra.

Diseño del programa

El primer paso es especificar, en términos generales, un algoritmo que describa el funcionamiento del programa, y que es el siguiente:

Presentar al usuario las instrucciones iniciales.
Poner en marcha el marcador y la puntuación posible.
Repetir el proceso:

> Generar los números y operaciones necesarios para una cuenta.
> Mostrar la cuenta.
> Introducir una respuesta.
> Si la respuesta ha sido correcta,
>
> > actualizar el marcador;
> > en caso contrario, presentar en pantalla la respuesta correcta.

Hasta que el usuario no desee continuar.
Mostrar la puntuación final.

En la segunda etapa del proceso de diseño se expresan con más detalle algunos aspectos particulares del programa. Como hay que generar dos números, lo mejor es crear un subprograma que se encargue de ello. El subprograma devuelve un entero positivo generado al azar y comprendido entre 1 y 100. Para simplificar la substracción y la división, el segundo número vuelve a generarse si resulta ser superior al primero.

La operación de la cuenta se elige generando otro entero aleatorio comprendido entre 1 y 4 y transfiriendo el control a uno de cuatro subprogramas que muestran la cuenta en pantalla, calculan el resultado de la misma (en forma de cociente y resto en el caso de la división), aceptan la introducción de una respuesta y la comprueban.

Como resultado, remiten una variable de verificación que indica si la respuesta del usuario es correcta o no.

Dado que no hay que crear ninguna estructura de datos, el programa ya está especificado con suficiente detalle y puede pasarse a la fase de escritura.

Variables

- **S** Puntuación del marcador.
- **P** Puntuación posible.
- **A,B** Números de la cuenta.
- **Q** Resultado calculado.
- **R** Resto de la división.
- **U** Respuesta del usuario.
- **V** Resto del usuario (en la división).
- **X** Parámetro devuelto por el módulo generador de números aleatorios (en el intervalo 1-100) para ser usado en la cuenta.
- **D** Parámetro devuelto por el módulo generador de números aleatorios (en el intervalo 1-4) para decidir la operación.
- **F** Variable de verificación: **F** = 1 si la respuesta del usuario es correcta, y **F** = 0 en caso contrario.
- **Z** Valor introducido al principio por el usuario para lanzar el generador de números aleatorios.
- **R$** Respuesta a la pregunta.

Diagrama de flujo

La figura 23.1 ilustra el flujo de control del programa ejemplo 23.1. No se incluyen aquí los módulos de substracción y multiplicación, porque su estructura es idéntica a la del de adición.

Programa

```
100 REM PROGRAMA EJEMPLO 23.1
105 REM EJERCICIOS DE ARITMETICA PRACTICA
110 REM
200 REM VISUALIZACION INICIAL DE LAS INSTRUCCIO
        NES DEL USUARIO
205 PRINT "ESTE PROGRAMA LE DARA PRACTICA A LA
          HORA DE HACER CALCULOS"
210 PRINT
215 PRINT "ESCRIBA LA RESPUESTA A CADA OPERACION
          CUANDO SE VISUALICE"
220 PRINT
225 PRINT "TOMESE EL TIEMPO QUE QUIERA, PERO
          DEBE TENER"
```

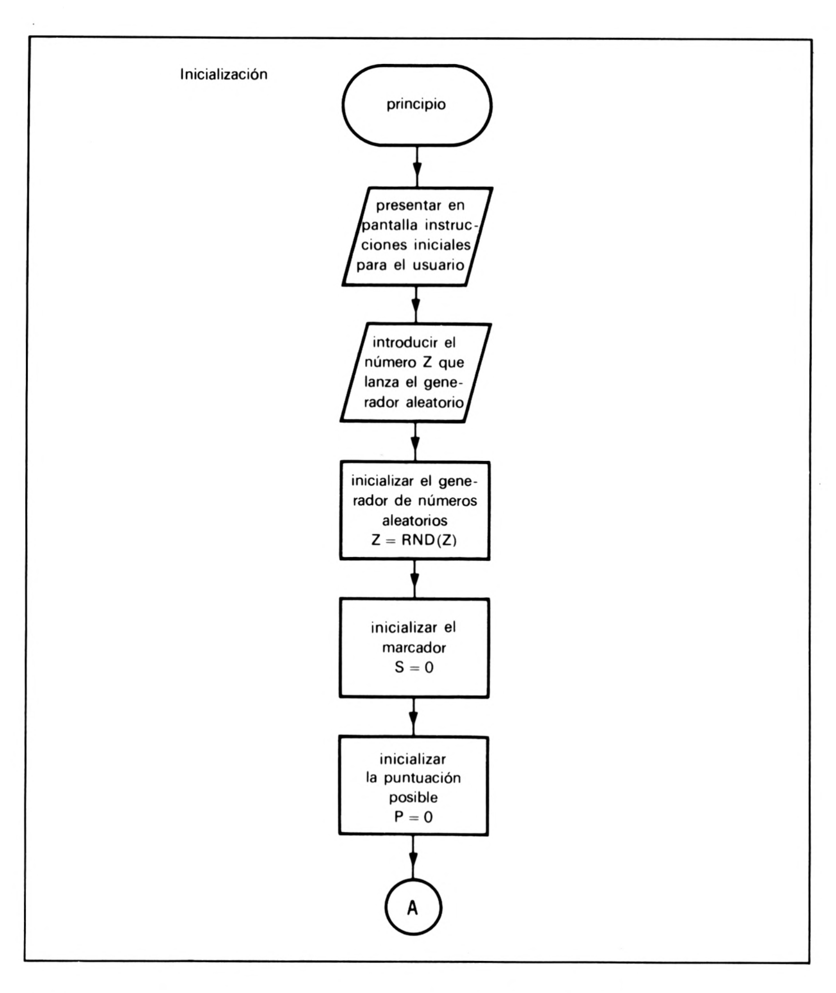
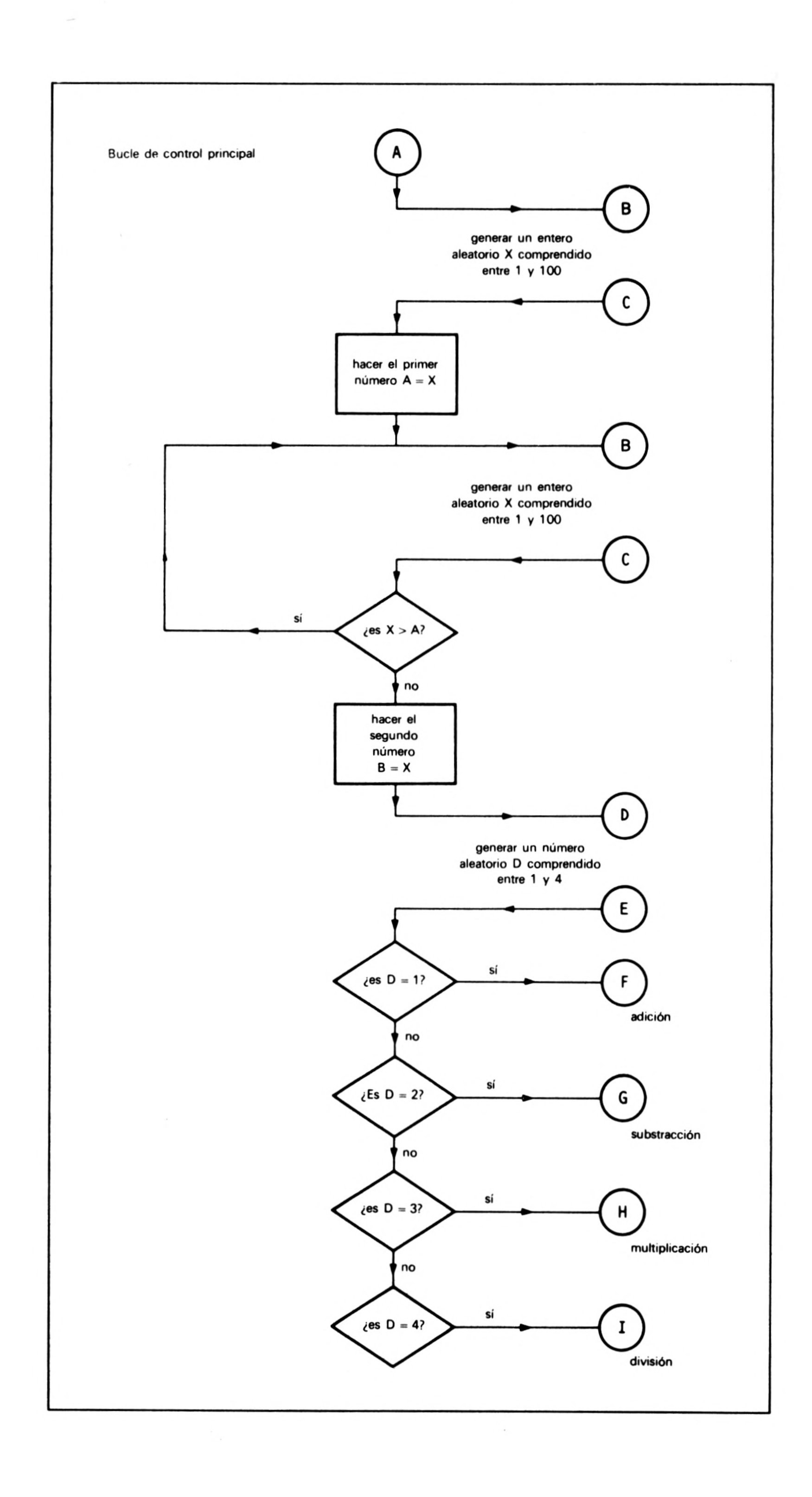
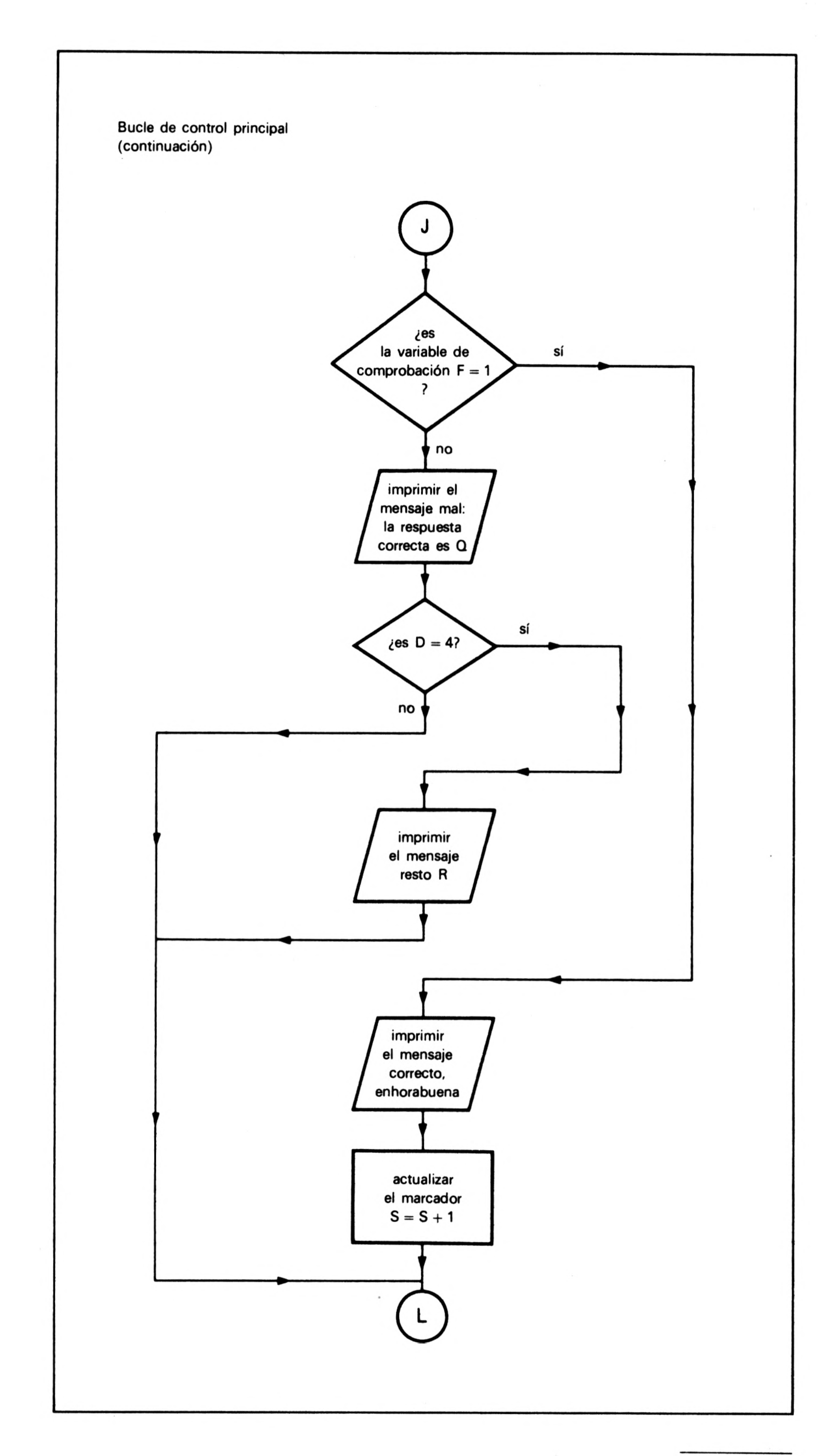
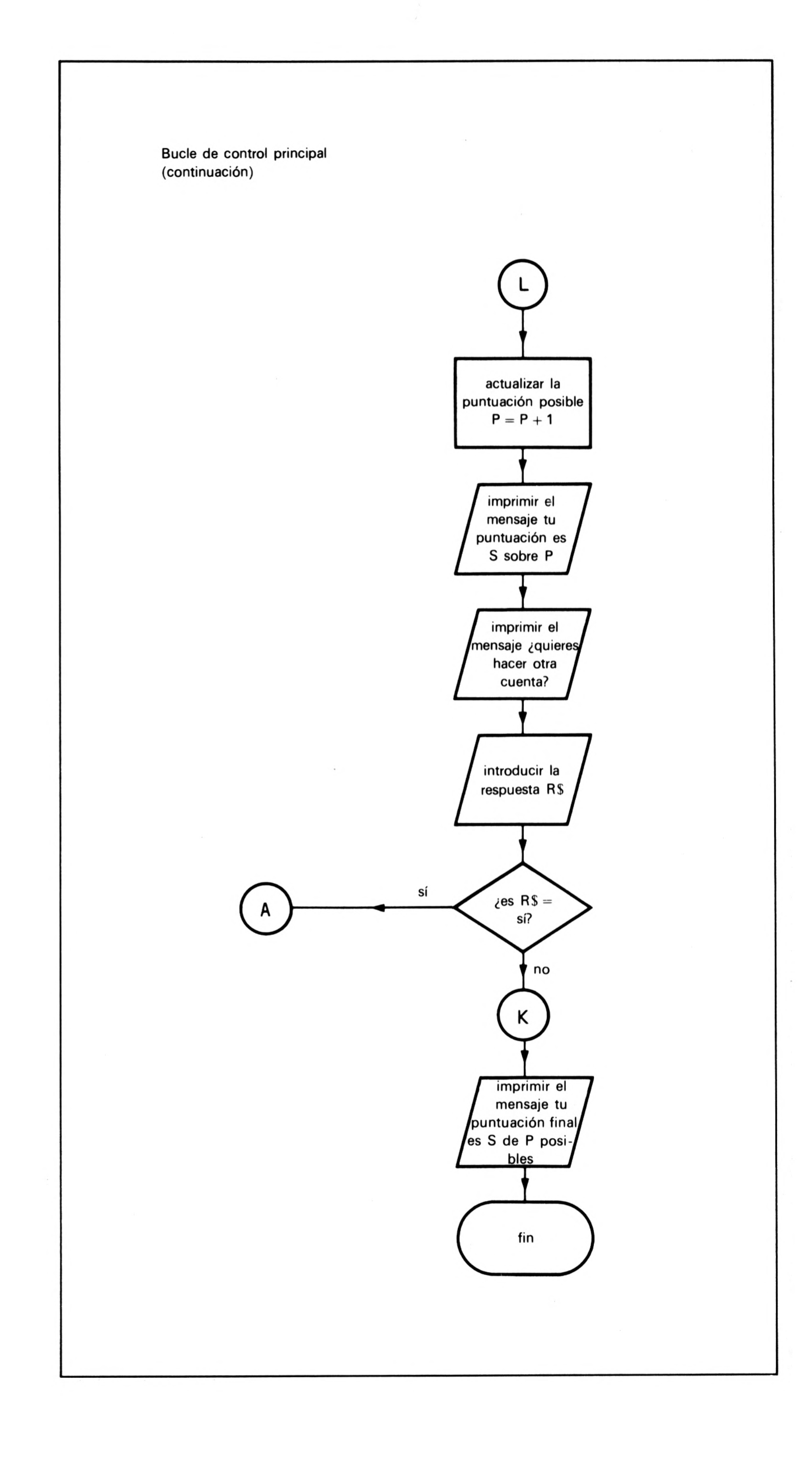


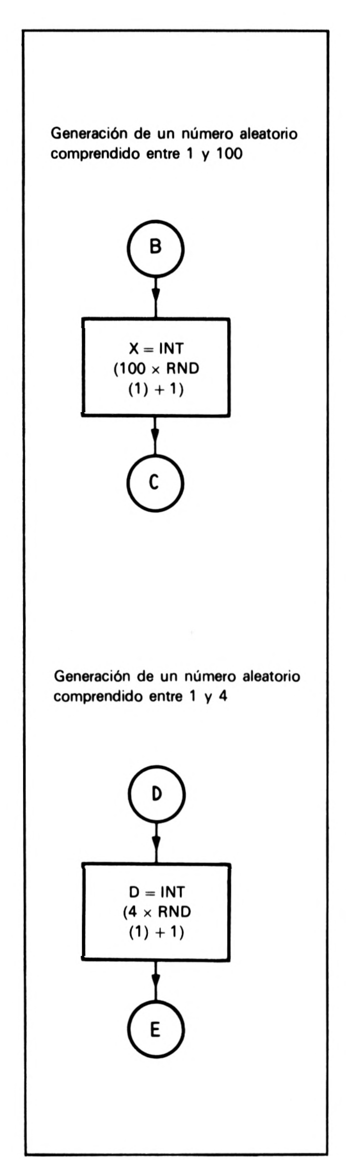
Figura 23.1
Diagrama de flujo del programa ejemplo 23.1

```
230 PRINT "LA RESPUESTA CORRECTA LA PRIMERA VEZ
           PARA OBTENER UN PUNTO"
235 PRINT
240 PRINT "SI TIENE UNA DIVISION, ESCRIBA PRIME
           RO EL COCIENTE,"
245 PRINT "LUEGO EL RESTO"
250 PRINT
255 PRINT "PARA COMENZAR EL PROGRAMA, ESCRIBA SU
           NUMERO DE LA SUERTE"
260 PRINT "DE HOY";
265 INPUT Z
270 LET Z = RND(-ABS(Z))
275 REM INICIALIZACION DEL MARCADOR Y POSIBLE
        TANTEO
280 LET S = 0
285 LET P = 0
290 PRINT
295 REM
```


Bucle de control principal
A
B
generar un entero aleatorio X comprendido entre 1 y 100
C
hacer el primer número A = X
B
generar un entero aleatorio X comprendido entre 1 y 100
C
sí
¿es X > A?
no
hacer el segundo número B = X
D
generar un número aleatorio D comprendido entre 1 y 4
E
¿es D = 1?
sí
F
adición
no
¿Es D = 2?
sí
G
substracción
no
¿es D = 3?
sí
H
multiplicación
no
¿es D = 4?
sí
I
división


Bucle de control principal
(continuación)
J
¿es
la variable de
comprobación F = 1
?
sí
no
imprimir el
mensaje mal:
la respuesta
correcta es Q
¿es D = 4?
sí
no
imprimir
el mensaje
resto R
imprimir
el mensaje
correcto,
enhorabuena
actualizar
el marcador
S = S + 1
L


Bucle de control principal
(continuación)
L
actualizar la puntuación posible P = P + 1
imprimir el mensaje tu puntuación es S sobre P
imprimir el mensaje ¿quieres hacer otra cuenta?
introducir la respuesta R$
¿es R$ = sí?
sí
A
no
K
imprimir el mensaje tu puntuación final es S de P posi-bles
fin



Subprograma de adición

F

presentar en pantalla la suma A + B =

calcular la respuesta Q = A + B

introducir la respuesta del usuario U

¿es U = Q?

sí

no

hacer la variable de verificación de respuesta F = 0

J

hacer la variable de verificación de respuesta F = 1

J

Bucle principal de control

```
300 REM BUCLE DE CONTROL PRINCIPAL
305 GOSUB 500: REM GENERACION DEL PRIMER NUMERO
310 LET A = X
315 GOSUB 500: REM GENERACION DEL SEGUNDO NUMERO
320 IF X > A THEN 315
325 LET B = X
330 GOSUB 550: REM GENERACION DE UN ENTERO COM
                   PRENDIDO ENTRE 1 Y 4
```

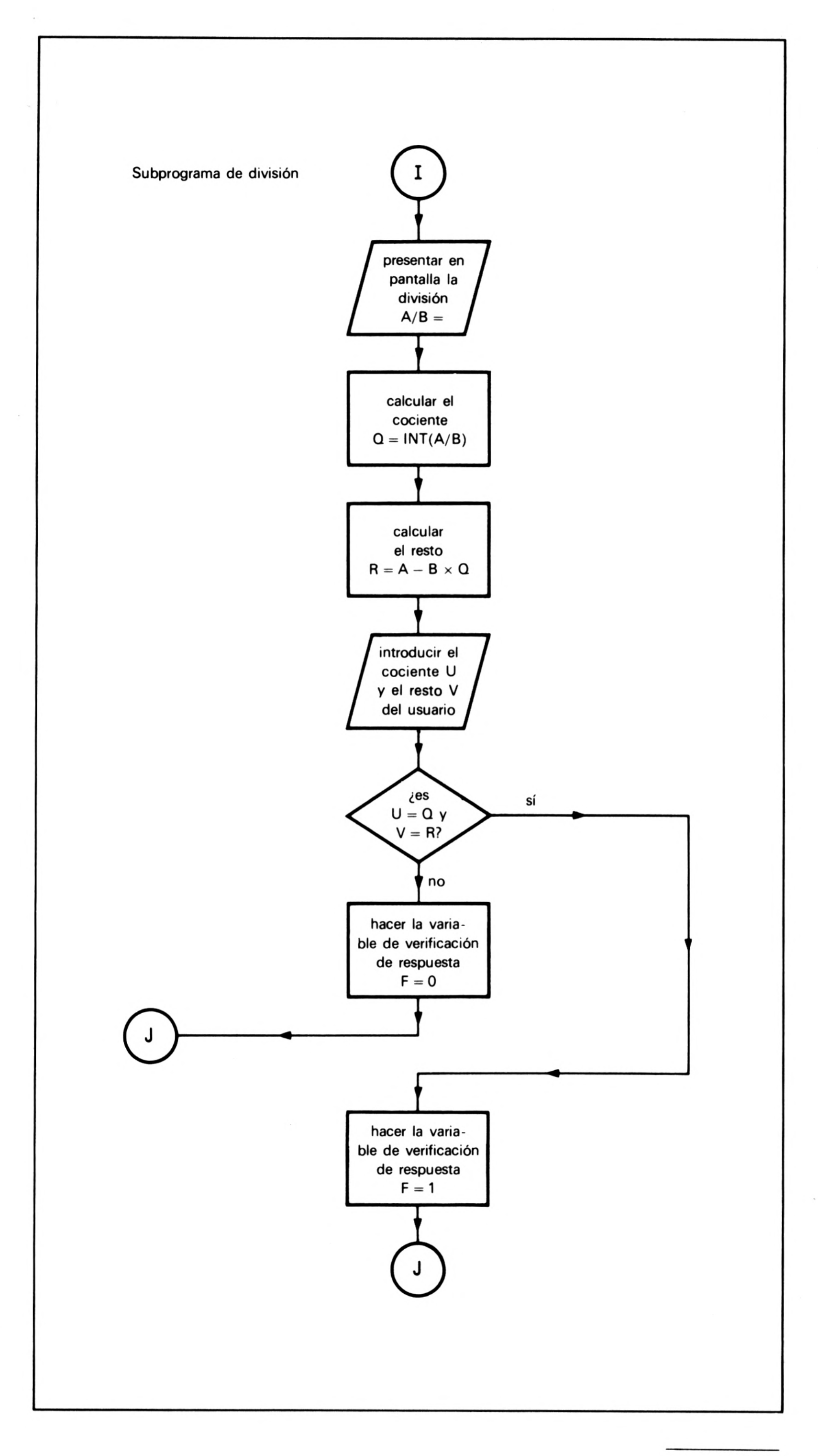
```
335 ON D GOSUB 600, 700, 800, 900
340 REM VISUALIZACION DE LA SUMA, INTRODUCCION Y
        COMPROBACION DE LA RESPUESTA
345 IF F = 1 THEN 385
350 PRINT
355 PRINT "MAL. LA RESPUESTA CORRECTA ES"; Q;
360 IF D = 4 THEN 375
365 PRINT
370 GOTO 395
375 PRINT "RESTO"; R
380 GOTO 395
385 PRINT "CORRECTO. ENHORABUENA"
390 LET S = S + 1
395 LET P = P + 1
400 PRINT
405 PRINT "SUS PUNTOS SON"; S; "DE"; P; "POSI
           BLES"
410 PRINT
415 PRINT "QUIERE HACER OTRA CUENTA?";
420 INPUT R$
425 IF R$ = "SI" THEN 300
430 REM
435 PRINT
440 PRINT "SU PUNTUACION FINAL ES"; S; "DE"; P;
          "PUNTOS POSIBLES"
445 PRINT
450 STOP
```

Subprogramas generadores de números aleatorios

```
500 REM GENERADOR ALEATORIO DE NUMEROS ENTEROS
        COMPRENDIDOS ENTRE 1 Y 100
505 LET X = INT(100*RND(1) + 1)
510 RETURN
550 REM GENERADOR ALEATORIO DE NUMEROS ENTEROS
        COMPRENDIDOS ENTRE 1 Y 4
555 LET D = INT(4*RND(1) + 1)
560 RETURN
```

Subprograma de adición

```
600 REM SUBPROGRAMA DE ADICION
605 PRINT A; "+"; B; "=";
610 LET Q = A + B
615 INPUT U
620 IF U = Q THEN 635
625 LET F = 0
630 RETURN
635 LET F = 1
640 RETURN
```


Subprograma de división
I
presentar en pantalla la división A/B =
calcular el cociente Q = INT(A/B)
calcular el resto R = A – B × Q
introducir el cociente U y el resto V del usuario
¿es U = Q y V = R?
sí
no
hacer la variable de verificación de respuesta F = 0
J
hacer la variable de verificación de respuesta F = 1
J

Subprograma de substracción

```
700 REM SUBPROGRAMA DE SUBSTRACCION
705 PRINT A; "-"; B; "=";
710 LET Q = A - B
715 INPUT U
720 IF U = Q THEN 735
725 LET F = 0
730 RETURN
735 LET F = 1
740 RETURN
```

Subprograma de multiplicación

```
800 REM SUBPROGRAMA DE MULTIPLICACION
805 PRINT A; "*"; B; "=";
810 LET Q = A * B
815 INPUT U
820 IF U = Q THEN 835
825 LET F = 0
830 RETURN
835 LET F = 1
840 RETURN
```

Subprograma de división

```
900 REM SUBPROGRAMA DE DIVISION
905 PRINT A; "/"; B; "=";
910 LET Q = INT(A/B)
915 LET R = A - Q * B
920 INPUT U, V
925 IF U = Q AND V = R THEN 940
930 LET F = 0
935 RETURN
940 LET F = 1
945 RETURN
1000 END
```

Puntos de interés

- Las instrucciones iniciales y los mensajes están escritos a un nivel adecuado al supuesto usuario, en este caso un niño.
- Los cuatro subprogramas de operaciones contienen ciertas instrucciones duplicadas, pero, dado que la división es un caso especial, esa repetición simplifica la estructura del programa.
- El "número de la suerte" que se introduce al principio sirve para lanzar el generador de números aleatorios.

23.4
Conclusión

Hemos expuesto en este capítulo algunas características generales de los programas interactivos y además hemos presentado un programa con esas características. La conclusión más importante es que los programas interactivos tienen que ser "**amistosos**", es decir, tener instrucciones, indicaciones y mensajes adecuados a las posibilidades del usuario al que van dirigidos. Además, deben ir siempre acompañados de una documentación para el usuario, de la mejor calidad.

23
Ejercicio

1. Resuma las características de un programa interactivo.
2. Aplique al programa ejemplo 23.1 algunas de las modificaciones sugeridas a continuación o todas ellas. Tenga en cuenta las técnicas de modificación de programas expuestas en el capítulo 14.
   a) Haga que el usuario pueda elegir el **nivel de dificultad** de las cuentas. Clasifique los niveles según una escala adecuada, de 1 a 10, por ejemplo. El nivel de dificultad determina el intervalo en el que están comprendidos los números usados en las operaciones. El nivel inferior trabaja dentro de un intervalo reducido de enteros positivos, mientras que el superior abarca incluso números negativos y fracciones. El usuario ha de poder cambiar de nivel de dificultad al término de cada operación.
   b) Haga que la división tenga la mitad de probabilidades de aparecer que las demás operaciones.
   c) Amplíe los cálculos para incluir en ellos tres o más números.
   d) Amplíe las operaciones para incluir determinaciones sencillas de porcentajes.
3. Redacte la documentación para el programador y la guía para el usuario correspondientes al programa ejemplo 23.1 o a la versión modificada según las sugerencias de la pregunta anterior.
4. Vuelva a escribir el programa ejemplo 23.1 utilizando la construcción **IF ... THEN ... ELSE** y agrupando varias instrucciones en una misma línea.
5. El juego de las **tres en raya** puede entenderse de la siguiente manera: el tablero de juego contiene una serie de **líneas ganadoras;** si las casillas y estas líneas ganadoras se numeran como ilustra la figura 23.2, a cada una de las primeras corresponderán las líneas indicadas a continuación:

Casilla	*Líneas ganadoras*			
1	1	4	7	
2	1	5		
3	1	6	8	
4	2	4		
5	2	5	7	8
6	2	6		
7	3	4	8	
8	3	5		
9	3	6	7	

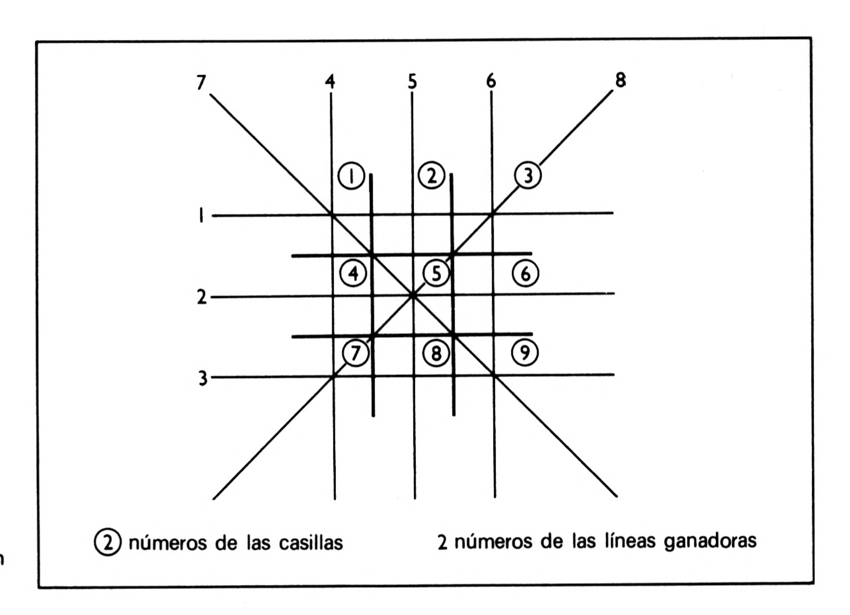


Figura 23.2
Líneas ganadoras en las tres en raya

El objetivo del juego es ocupar una línea ganadora completa. En cualquier fase de una partida, cada una de las líneas ofrece más o menos ventajas a un jugador que al otro. Así, para el jugador identificado mediante el símbolo 0, el orden de ventaja sería:

Máxima: líneas ganadoras con dos fichas 0,
líneas ganadoras con una ficha 0,
líneas ganadoras vacías o con dos fichas distintas,
líneas ganadoras con una ficha X.
Mínima: líneas ganadoras con dos fichas X.

La siguiente estrategia permite escoger la casilla óptima en cada jugada:

Si en una línea hay dos fichas del oponente y una casilla vacía, ocuparla y ganar.
Si en una línea hay dos fichas del oponente y ninguna del que juega, ocupar la casilla vacía para impedir la victoria del contrario.
En caso contrario, ir ocupando las líneas con una ficha propia y ninguna del oponente.
En caso contrario, bloquear al oponente todas las líneas empezadas por él que sea posible y empezar también todas las líneas posibles.

La estrategia expuesta puede materializarse en un programa en el que el ordenador juegue contra el usuario. En cada turno, el ordenador estudiará todas las casillas libres a la luz de la estrategia descrita y ocupará la que resulte más ventajosa.

Utilice las estructuras de datos adecuadas para representar la situación del juego en cada momento y para almacenar la relación entre casillas y líneas ganadoras.


RESET
RETURN

24 Gráficos

¿Qué es la belleza?
Nadie lo sabe todavía, porque es demasiado evidente.

SALVADOR DALÍ

Este capítulo se aparta del mundo exacto y relativamente seguro de la informática comercial y científica para adentrarse en aguas no del todo sondeadas y en regiones que, al decir de algunas personas, los ordenadores jamás debieron haber hollado: hablamos, en definitiva, del mundo del arte, en el que lo intuitivo, lo trascendental y lo subconsciente dominan sobre la lógica y la precisión.

Pese a todas esas reservas, hay ya una experiencia considerable en la generación directa o mediatizada de obras de arte con ordenador; por su parte, la arquitectura y la ingeniería se sirven cada vez más de las técnicas de diseño apoyado por ordenador. La programación y el arte tienen mucho en común, sobre todo porque ambas actividades obligan a plasmar una idea abstracta en un objeto concreto.

En este capítulo consideraremos al ordenador como un "pincel mágico": pincel porque necesita una mano que lo guíe para producir imágenes visibles, y mágico porque con algunas técnicas de programación pueden crearse efectos aleatorios o inesperados. Se expondrán

algunas técnicas sencillas de trazado de gráficos y se escribirán otros tantos programas que las aplicarán en la práctica.

En este campo el diseño y la estructuración de programas adquieren una importancia vital, ya que contribuyen a superar una de las principales dificultades planteadas por el Basic: la dependencia del ordenador.

24.1

Dependencia de los recursos gráficos respecto del ordenador

La provisión de recursos gráficos centra una parte importante de la competencia entre fabricantes de microordenadores, lo que hace que los ofrecidos por las diversas marcas y modelos sean muy diversos.

Teniendo esto en cuenta, los programas, ejemplos y gráficos de este capítulo están escritos para una máquina genérica, que se ha pretendido sea lo más general posible, de tal manera que los programas sean fácilmente adaptables a cualquier máquina.

24.2

El lienzo de rayos catódicos

El lienzo sobre el que se harán los gráficos es la pantalla del ordenador. Aunque cuando está vacía parece una superficie uniforme, hay que imaginarla como una cuadrícula de pequeños cuadrados organizados en filas y columnas, como ilustra la figura 24.1.

El tamaño de cada casilla y, por tanto, el número de ellas que tiene la pantalla, da la **resolución** del sistema de gráficos disponible. Cuando cada cuadrado es igual al área ocupada por un carácter durante la visualización se habla de baja resolución, y de **alta resolución** cuando los espacios son considerablemente menores que el área ocupada por el carácter.

Los gráficos de este capítulo se ejecutarán a baja resolución, pero en realidad las técnicas utilizadas para crearlos son independientes de esa característica. Además, muchos ordenadores disponen de una serie de **caracteres gráficos** que pueden combinarse para proporcionar la ilusión de una resolución más elevada (véase la figura 24.2).

Para determinar la posición de cada uno de los espacios o casillas de la pantalla, se numeran las filas y columnas de los mismos y se usan esos valores como coordenadas (véase la figura 24.3), lo que significa que puede utilizarse la **geometría analítica** para generar gran diversidad de formas. De todas maneras, este capítulo no presupone conocimiento alguno de geometría analítica por parte del lector, aunque en el ejercicio propuesto al final del mismo se han planteado unas pocas

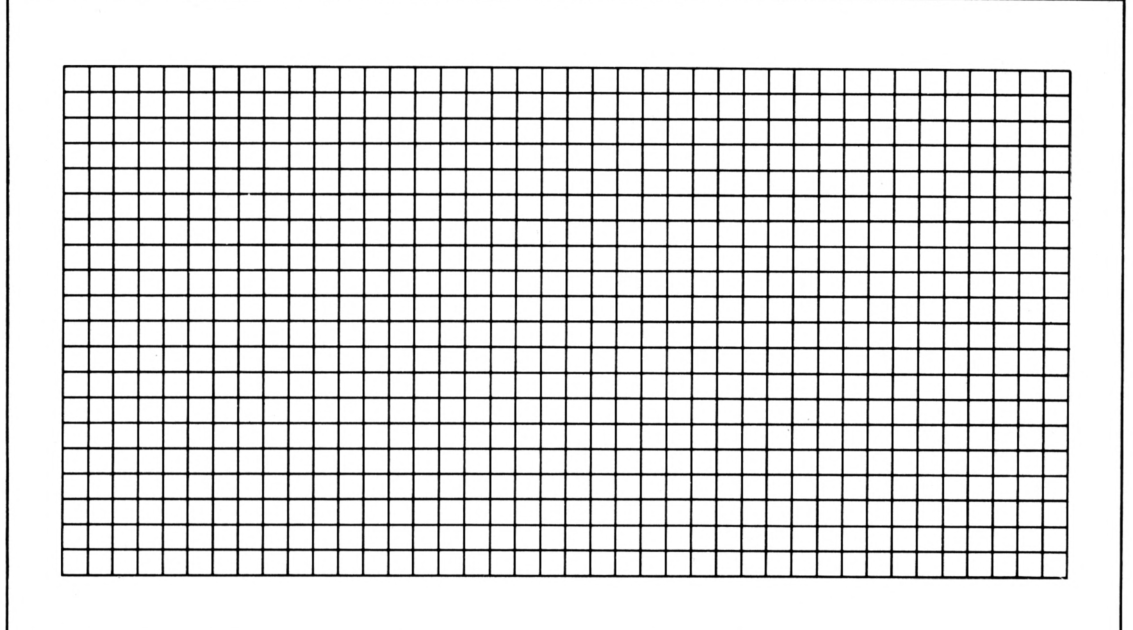
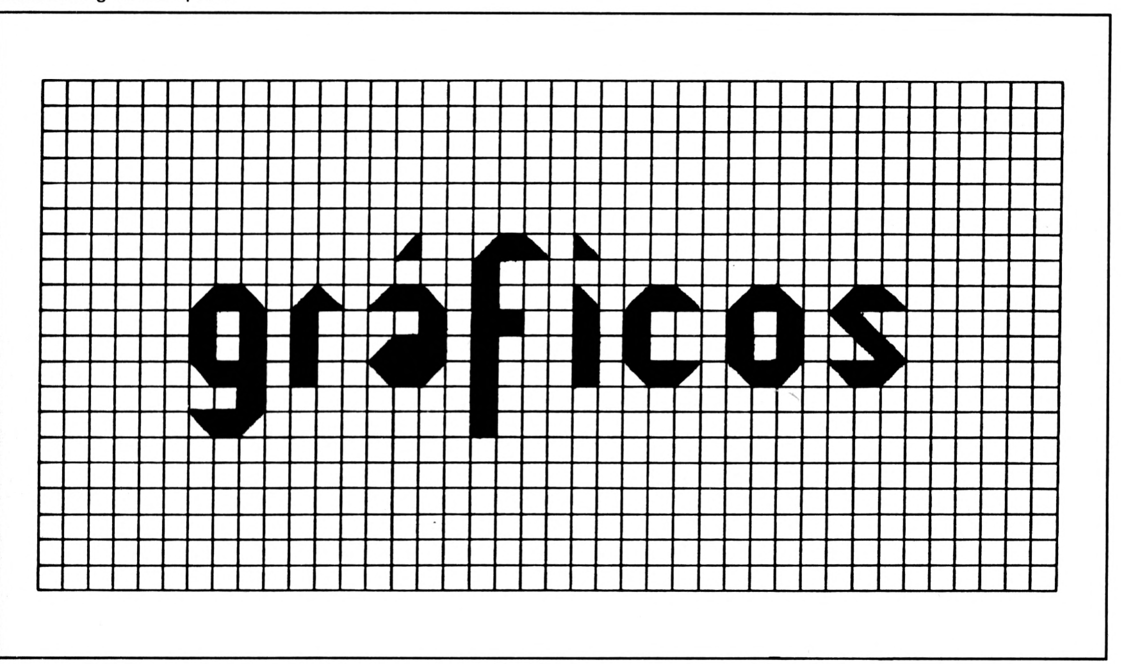


Figura 24.1
La cuadrícula de la pantalla

Figura 24.2
Caracteres gráficos típicos



preguntas que permitirán, a quienes los posean, sacar partido a dichos conocimientos.

A partir de ahora supondremos que la pantalla tiene **A** caracteres de anchura y **H** de altura. Las columnas se numerarán de 0 a **A**-1 y las filas de 0 a **H**-1. Los valores de estas variables se fijan al principio del programa con arreglo al modelo de ordenador que se use; en el caso de la versión elegida aquí, **A** vale 40 y **H** 20.

Comparar la pantalla de rayos catódicos con un lienzo no es muy correcto, porque los gráficos creados en la primera pueden moverse. El segundo de los programas ejemplo propuestos constituye una aplicación sencilla de esa posibilidad.

24.3

Pinceladas en Basic

Igual que un cuadro es el resultado de un número determinado de pinceladas individuales, un gráfico se genera a partir de operaciones de programación sencillas. Las más elementales de éstas son: borrar la pantalla, escribir un carácter o un símbolo en una posición determinada de la pantalla y leer el carácter situado en una posición determinada de la pantalla. Las "pinceladas" más complejas —como tender una recta entre dos puntos dados o sombrear cierta zona rectangular— pueden describirse en términos de las tres operaciones elementales mencionadas.

En las próximas tres secciones veremos cómo llevar a la práctica dichas operaciones. Ya se ha dicho que dependen del ordenador en uso y que los subprogramas encargados de realizarlas se escriben en la versión del lenguaje elegida. Todos estos subprogramas se utilizarán algo más adelante en este mismo capítulo. Están escritos para un sistema de gráficos en blanco y negro, pero resulta muy fácil ampliarlos para incluir en ellos el color.

24.4

Borrar la pantalla

Al comienzo de cualquier programa de gráficos y en ciertos momentos de los programas gráficos con animación, es preciso borrar la pantalla completa. En la mayor parte de las versiones del Basic, y la elegida aquí es una de esas, esto se consigue con una sola instrucción.

Subprograma

```
9000 REM LIMPIEZA DE LA PANTALLA
9020 CLS
9030 RETURN
```

Escribir un carácter en la pantalla

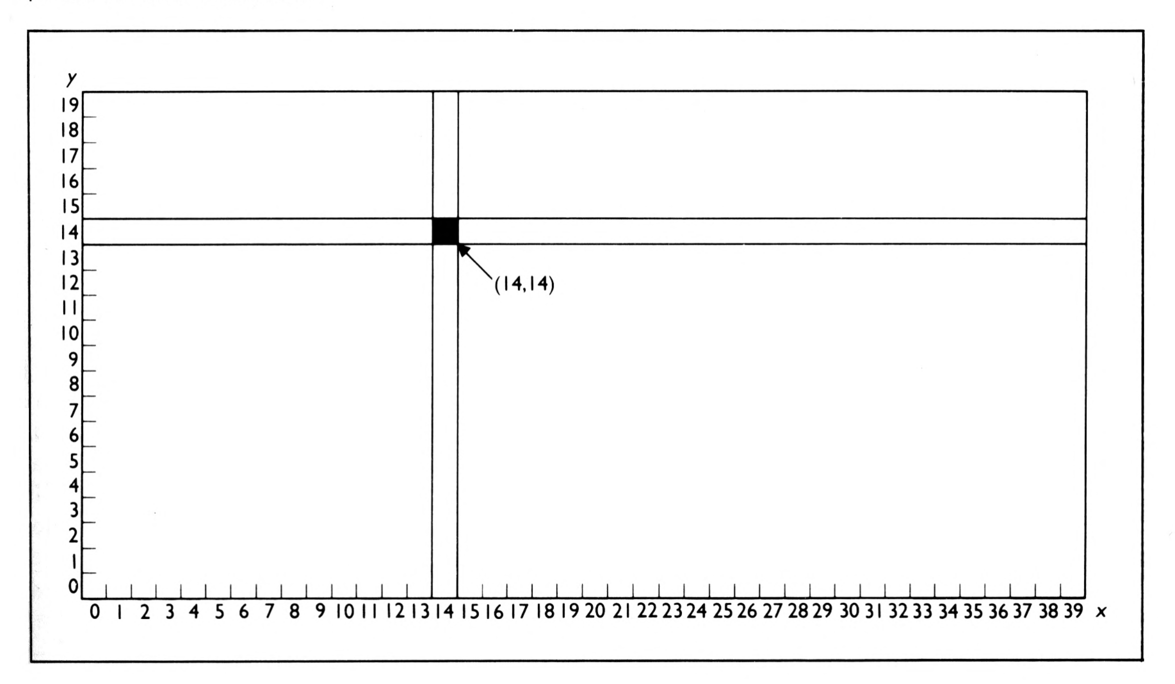
Un subprograma para escribir un carácter en una posición determinada de la pantalla necesita como parámetros las coordenadas *X* e *Y* de esa posición y, si se trabaja en color, el que desea utilizarse.

Se plantea un primer problema cuando las coordenadas *X* e *Y* caen fuera de las dimensiones de la pantalla. Para solucionarlo utilizaremos el subprograma listado a continuación.

Subprograma

```
9100 REM ESCRITURA DE UN CARACTER DE LA PANTALLA
9110 REM PARAMETROS
9115 REM X, Y: COORDENADAS DE LA POSICION DEL
         CARACTER
9120 REM C$: CARACTER
9125 REM CRUCE
9130 LET X1 = X - INT(X/A) * A
9135 LET Y1 = Y - INT(Y/H) * H
9140 LOCATE X1 * 2, Y1 * 3: PRINT C$
9145 RETURN
```

Figura 24.3
Señalización de la superficie de la pantalla mediante coordenadas



24.6

Leer un carácter en la pantalla

Un subprograma capaz de leer el carácter situado en una posición dada de la pantalla necesita como parámetros las coordenadas de esa posición y entrega como resultado el carácter (más su color, en sistemas que lo tengan).

Como en el caso anterior, las coordenadas corresponden al esquema ilustrado en la figura 24.3.

Subprograma

```
9200 REM LECTURA DE UN CARACTER EN LA PANTALLA
9205 REM VERSION DEL BASIC IBM PC
9210 REM PARAMETROS
9215 REM X, Y: COORDENADAS DE LA POSICION DEL
         CARACTER
9220 REM C$: CARACTER
9225 REM CRUCE
9230 LET X1 = X - INT(X/A) * A
9233 LET Y1 = Y - INT(Y/H) * H
9235 LET C$ = CHR$(LOCATE(X1 * 2, Y1 * 3))
9245 RETURN
```

24.7

Programa ejemplo 24.1

El objetivo de este programa es dibujar en la pantalla ese conocido dibujo que puede interpretarse como dos caras enfrentadas o como una copa.

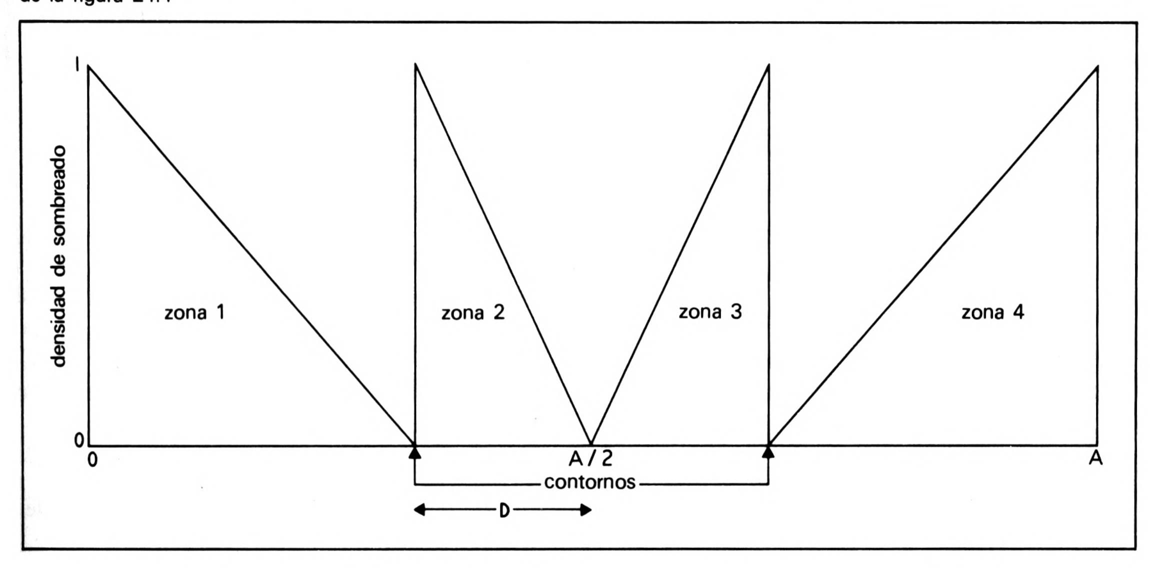
El programa utiliza los subprogramas de borrado de la pantalla y de escritura de caracteres que acabamos de estudiar, más un procedimiento de sombreado mediante números aleatorios que veremos a continuación.

La imagen es simétrica con respecto a una línea vertical que cruza la pantalla por el centro y queda definida por dos líneas de contorno, como se ve en la figura 24.4. El resto recibe un sombreado de intensidad variable, pero que sigue la misma pauta en todas las líneas horizontales. La densidad es máxima en los dos bordes externos de la pantalla y disminuye paulatinamente hasta hacerse nula en las dos líneas del contorno. En la imagen central —la copa— la densidad es máxima en los bordes y nula en la línea media de simetría. Estas variaciones de densidad a lo largo de una línea horizontal se ilustran en la figura 24.5.

Figura 24.4
Contornos del programa ejemplo 24.1

eje
de
simetría

Figura 24.5
Densidad de sombreado a lo largo de una de las líneas horizontales de la figura 24.4



Diseño del programa

La estructura general del programa es la siguiente:

Leer, en un conjunto de datos almacenados, las posiciones de los caracteres que conforman la línea de contorno.

Introducir cuatro caracteres, uno para cada una de las zonas de la imagen.

Borrar la pantalla.

Para cada línea de la figura, repetir el proceso:

- sombrear la zona 1, desde el límite izquierdo hasta el contorno izquierdo;
- sombrear la zona 2, desde el contorno izquierdo hasta el centro;
- sombrear la zona 3, desde el centro hasta el contorno derecho;
- sombrear la zona 4, desde el contorno derecho hasta el límite derecho.

Veamos ahora con más detalle los pasos enumerados.

Las líneas de contorno

Como las dos líneas son simétricas, basta especificar los datos correspondientes a una de ellas. La línea se define en términos de su desplazamiento **D** (véase la figura 24.5) con respecto al eje central de simetría. Cuando en una misma línea horizontal hay varios caracteres, basta con especificar el más externo, porque los demás se rellenarán durante el proceso de sombreado.

Sombreado con números aleatorios

En este método se considera la densidad del sombreado como la probabilidad de presentar en pantalla un carácter en una posición determinada. La densidad oscila entre cero (probabilidad nula) y uno (certeza completa).

Para determinar si en una posición dada aparecerá o no un carácter, se genera un número aleatorio entre 0 y 1: si es menor que la probabilidad de que aparezca, el carácter se lleva a la pantalla.

Dado que la densidad varía linealmente en cada una de las zonas, como se observa en la figura 24.5, puede usarse el teorema de los triángulos semejantes para determinar la probabilidad en cada posición. El cálculo para la zona 1 se ilustra en la figura 24.6, y se hace como sigue:

La anchura de la zona es **A/2 − D**.

Si **X** es la distancia desde la izquierda de la zona, la longitud del resto de la misma será **A/2 − D − X**.

Según el teorema de los triángulos semejantes,

$$\frac{\mathbf{P}}{\mathbf{A/2 - D - X}} = \frac{\mathbf{1}}{\mathbf{A/2 - D}}$$

donde **P** es la probabilidad de sombreado a la distancia **X**. En otras palabras:

$$\mathbf{P} = \frac{\mathbf{A/2 - D - X}}{\mathbf{A/2 - D}}$$

Para comprobar que la fórmula es correcta, basta hacer los cálculos para **X** = 0 y para **X** = **A/2** − **P**: en el primer caso **P** vale 1 y en el segundo, 0.

Los cálculos de las probabilidades de sombreo para las otras zonas se hacen de forma similar. El diseño del programa está suficientemente detallado y puede pasarse a la fase de escritura.

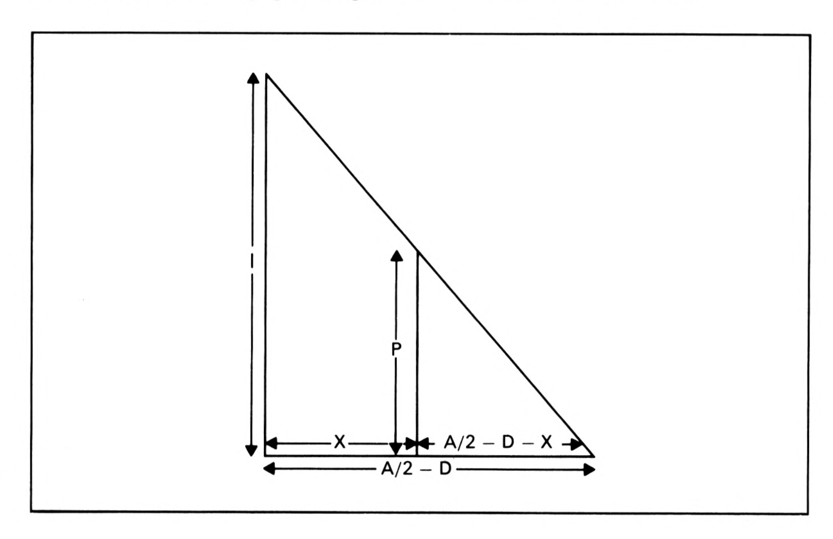


Figura 24.6
Cálculo de las probabilidades de sombreado

Variables

D(19)	Matriz de desplazamientos de los caracteres de la línea de contorno respecto al eje central.
C1$, C2$, C3$, C4$	Cuatro caracteres usados para sombrear.
A	Anchura total de la pantalla.
H	Altura de la pantalla.
X, Y	Coordenadas del carácter en curso y parámetros de los subprogramas.
C$	Carácter en curso y parámetro de subprograma.

P	Probabilidad de sombreado.
Z	Número aleatorio inicial.
A2	Anchura de la mitad de la pantalla.
D1	Desplazamiento del contorno en la línea en curso.

Diagrama de flujo

La figura 24.7 ilustra el flujo de control del bucle principal de mando del programa ejemplo 24.1.

Programa

```
1000 REM PROGRAMA EJEMPLO 24.1
1005 REM DIBUJO DE LAS CARAS/COPA
1010 REM
1100 REM ASIGNACION DE DIMENSIONES DE LA PANTALLA
1105 DIM D(19): REM 1 MENOS QUE LA ALTURA DE LA
                     PANTALLA
1110 LET H = 20: REM ALTURA DE LA PANTALLA
1115 LET A = 40: REM ANCHURA DE LA PANTALLA
1120 LET A2 = INT(A/2)
1125 REM
1200 REM INSTRUCCIONES INICIALES DEL USUARIO
1205 PRINT "DIBUJO DE LAS CARAS/COPA"
1210 PRINT
1215 REM LAS INSTRUCCIONES DEL USUARIO SE DEJAN
         COMO EJERCICIO
1220 REM
1300 REM LEER LOS DESPLAZAMIENTOS DEL CONTORNO
1305 FOR K = 0 TO 19
1310 READ D(K)
1315 NEXT K
1320 DATA 14, 13, 8, 9, 9
1325 DATA 8, 8, 9, 8, 5
1330 DATA 5, 5, 6, 7, 8
1335 DATA 7, 7, 8, 9, 10
1340 REM
1400 REM INTRODUCCION DE CARACTERES E INICIALIZA
         CION
1405 REM DEL GENERADOR DE NUMEROS ALEATORIOS
1410 PRINT "INTRODUCIR LOS CARACTERES QUE VAN A
            SER VISUALIZADOS"
1415 PRINT " EN LAS CUATRO ZONAS"
1420 INPUT C1$, C2$, C3$, C4$
1425 PRINT "INTRODUZCA UN NUMERO POSITIVO PARA
            INICIALIZAR"
1430 PRINT "EL GENERADOR DE NUMEROS ALEATORIOS"
1435 INPUT Z
1440 LET Z = RND(-Z)
1445 REM
1500 REM LIMPIAR PANTALLA
1505 GOSUB 9000
1510 REM
```

Bucle de control principal

```
1600 REM BUCLE PRINCIPAL DE CONTROL
1605 FOR Y = 0 TO H - 1
1610 LET D1 = D(Y)
1615 REM
1700 REM ZONA 1
1705 LET C$ = C1$
1710 FOR X = 0 TO A2 - D1 - 1
1715 LET P = (A2 - D1 - X)/(A2 - D1)
1720 IF RND(1) > P THEN 1730
1725 GOSUB 9100: REM ESCRITURA DEL CARACTER EN LA
                         PANTALLA
1730 NEXT X
1735 REM
1800 REM ZONA 2
1805 LET C$ = C2$
1810 FOR X = A2 - D1 TO A2 - 1
1815 LET P = (A2 - X)/D1
1820 IF RND(1) > P THEN 1830
1825 GOSUB 9100: REM ESCRITURA DEL CARACTER EN LA
                         PANTALLA
1830 NEXT X
1835 REM
1900 REM ZONA 3
1905 LET C$ = C3$
1910 FOR X = A2 TO A2 + D1 - 1
1915 LET P = (X - A2)/D1
1920 IF RND(1) > P THEN 1930
1925 GOSUB 9100: REM ESCRITURA DEL CARACTER EN LA
                         PANTALLA
1930 NEXT X
1935 REM
2000 REM ZONA 4
2005 LET C$ = C4$
2010 FOR X = A2 + D1 TO A - 1
2015 LET P = (X - A2 - D1)/(A2 - D1)
2020 IF RND(1) > P THEN 2030
2025 GOSUB 9100: REM ESCRITURA DEL CARACTER EN LA
                         PANTALLA
2030 NEXT X
2035 REM
2100 REM FIN DEL BUCLE DE CONTROL PRINCIPAL
2105 NEXT Y
2110 REM
9000 REM INCLUYE LIMPIEZA DE PANTALLA Y VISUALI
          ZACION DE LOS SUBPROGRAMAS DE CARACTERES
9100 REM
9999 END
```

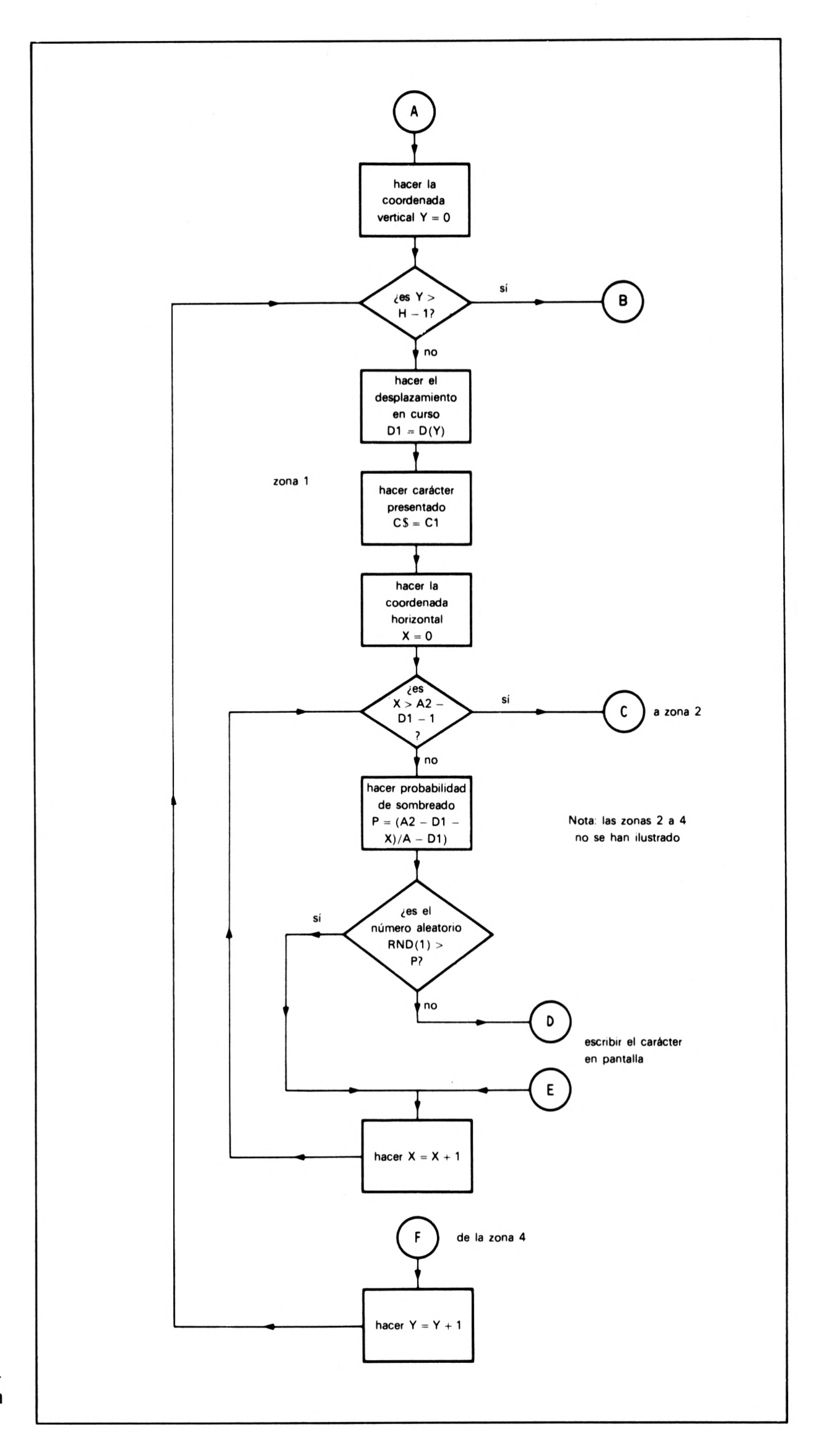


Figura 24.7
Diagrama de flujo del bucle principal de control del programa ejemplo 24.1

Puntos de interés

- El programa sigue la misma pauta para cada zona, pero a pesar de ello no pueden disponerse más bucles internos, porque la fórmula que calcula la probabilidad de sombreado es diferente en cada caso.
- La condición utilizada en cada una de las zonas es la de no presentación en pantalla del carácter, lo que simplifica la estructura del programa.
- Hay que incluir los subprogramas de borrado de pantalla y escritura de caracteres descritos al comienzo del capítulo.

24.8

Programa ejemplo 24.2

El objetivo de este programa es trazar en la pantalla una **trayectoria aleatoria.** En cada punto de una trayectoria de esta clase hay ciertas probabilidades de girar a la izquierda o a la derecha o de no girar, y la opción se escoge al azar en función de dichas probabilidades.

Las trayectorias aleatorias tienen aplicaciones en física y química, ya que se aproximan al comportamiento de los átomos o moléculas en ciertas condiciones.

Diseño del programa

Al principio del programa, el usuario debe indicar la probabilidad de que se produzca un giro tras cada paso, tras lo cual se asigna la mitad de la misma al giro a la izquierda y la otra mitad al giro a la derecha. También son necesarios dos caracteres para marcar el camino, que se superponen mutuamente si la trayectoria vuelve sobre sus pasos. Por último, hay que introducir una semilla para el generador de números aleatorios.

Las direcciones —arriba, derecha, abajo, izquierda— vienen representadas por los enteros comprendidos entre 0 y 3. Así, un giro a la derecha se obtiene sumando 1 a la dirección y saltando de 3 a 0 si fuese necesario. El giro a la izquierda se obtiene restando 1 a la dirección. Si la trayectoria sale de la pantalla, la instrucción de cruce de los límites contenida en los subprogramas de lectura y escritura solucionará la situación.

La estructura general del programa es la siguiente:

Presentar las instrucciones iniciales para el usuario.

Introducir la probabilidad de giro, los caracteres del camino y la semilla para el generador de números aleatorios.

Borrar la pantalla.

Presentar el primer carácter en el centro de la pantalla y llevar la dirección a cero (hacia arriba).

Repetir indefinidamente el proceso:

> utilizar un número al azar para decidir si hay que desviarse a derecha o izquierda o no hay que desviarse;
>
> actualizar las coordenadas de la posición en curso;
>
> determinar el carácter de la posición en curso y mostrar el otro en pantalla.

Las especificaciones del programa están ya suficientemente detalladas y aquél puede escribirse.

Variables

P	Probabilidad de giro a izquierda o derecha.
D	Dirección en curso.
C1$, C2$	Caracteres del camino.
C$	Carácter en curso y parámetro del subprograma.
X, Y	Coordenadas de la posición del carácter en curso y parámetros de subprograma.
A	Anchura de la pantalla.
H	Altura de la pantalla.
Z	Inicializador del generador de números aleatorios.
R	Número aleatorio en curso.

Diagrama de flujo

La figura 24.8 ilustra el flujo de control del bucle principal de mando del programa ejemplo 24.2.

Programa

```
1000 REM PROGRAMA EJEMPLO 24.2
1005 REM VISUALIZACION DE UNA TRAYECTORIA ALEATO
         RIA
1010 REM
1100 REM ASIGNACION DE DIMENSIONES DE LA PANTALLA
1105 LET A = 40: REM ANCHURA DE LA PANTALLA
1110 LET H = 20: REM ALTURA DE LA PANTALLA
1115 REM
```

```
1200 REM INSTRUCCIONES INICIALES DEL USUARIO
1205 PRINT "TRAYECTORIA ALEATORIA"
1210 PRINT
1215 REM SE DEJAN COMO EJERCICIO LAS INSTRUCCIO
         NES DEL USUARIO
1220 REM
1300 REM INTRODUCIR LA PROBABILIDAD DE GIRO, LOS
         CARACTERES DEL CAMINO,
1305 REM E INICIALIZAR EL GENERADOR DE NUMEROS
         ALEATORIOS
1310 PRINT "PROBABILIDAD DE UN GIRO TRAS CADA
            PASO?"
1315 INPUT P
```

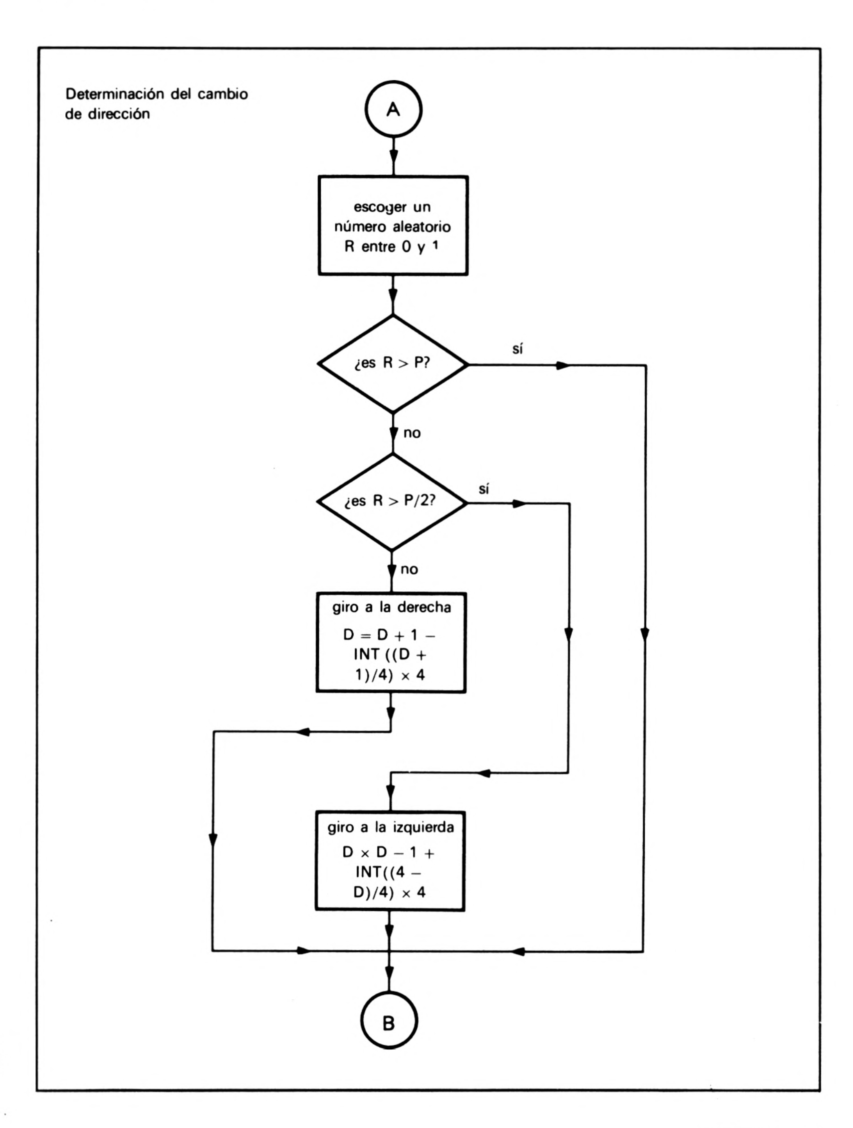
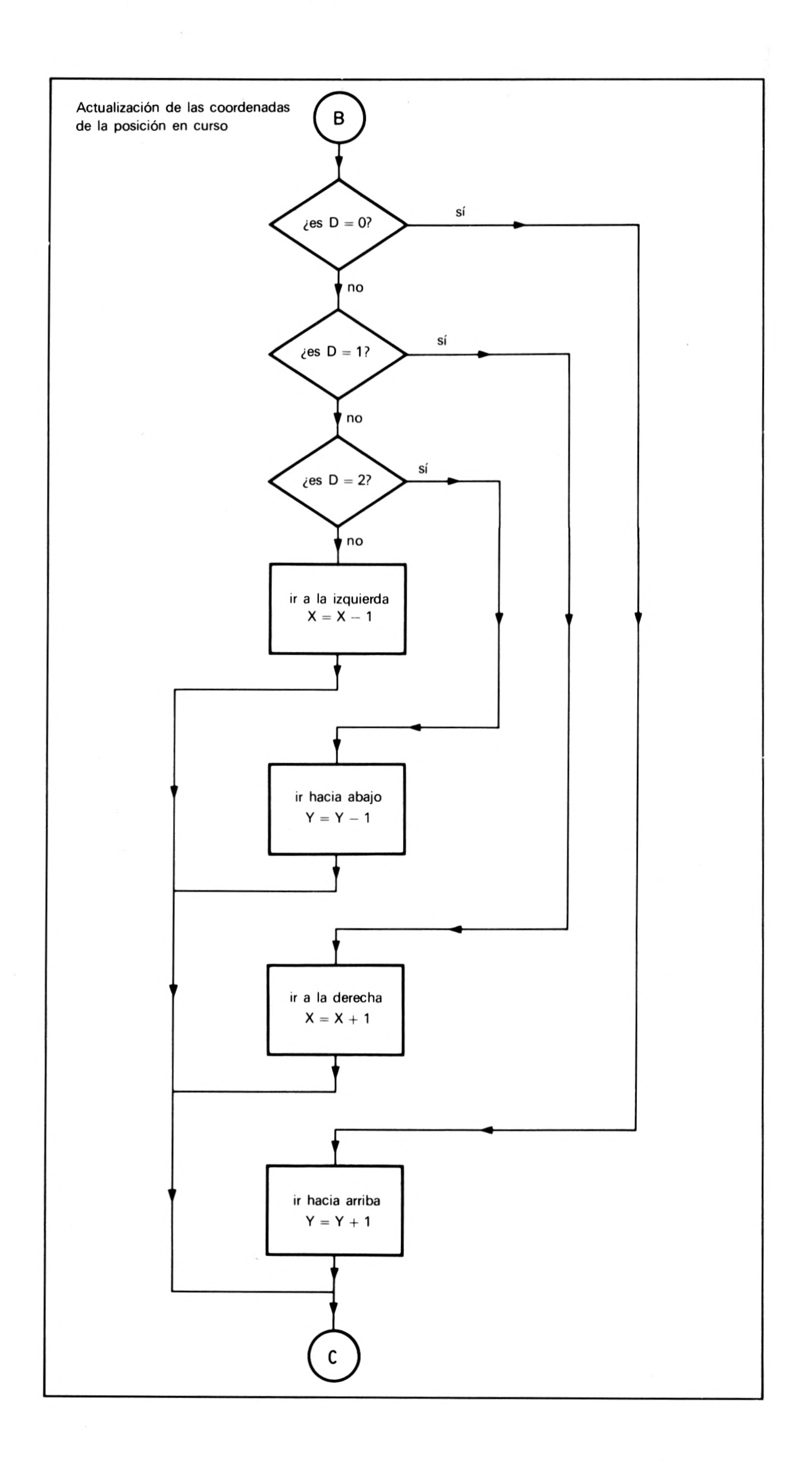
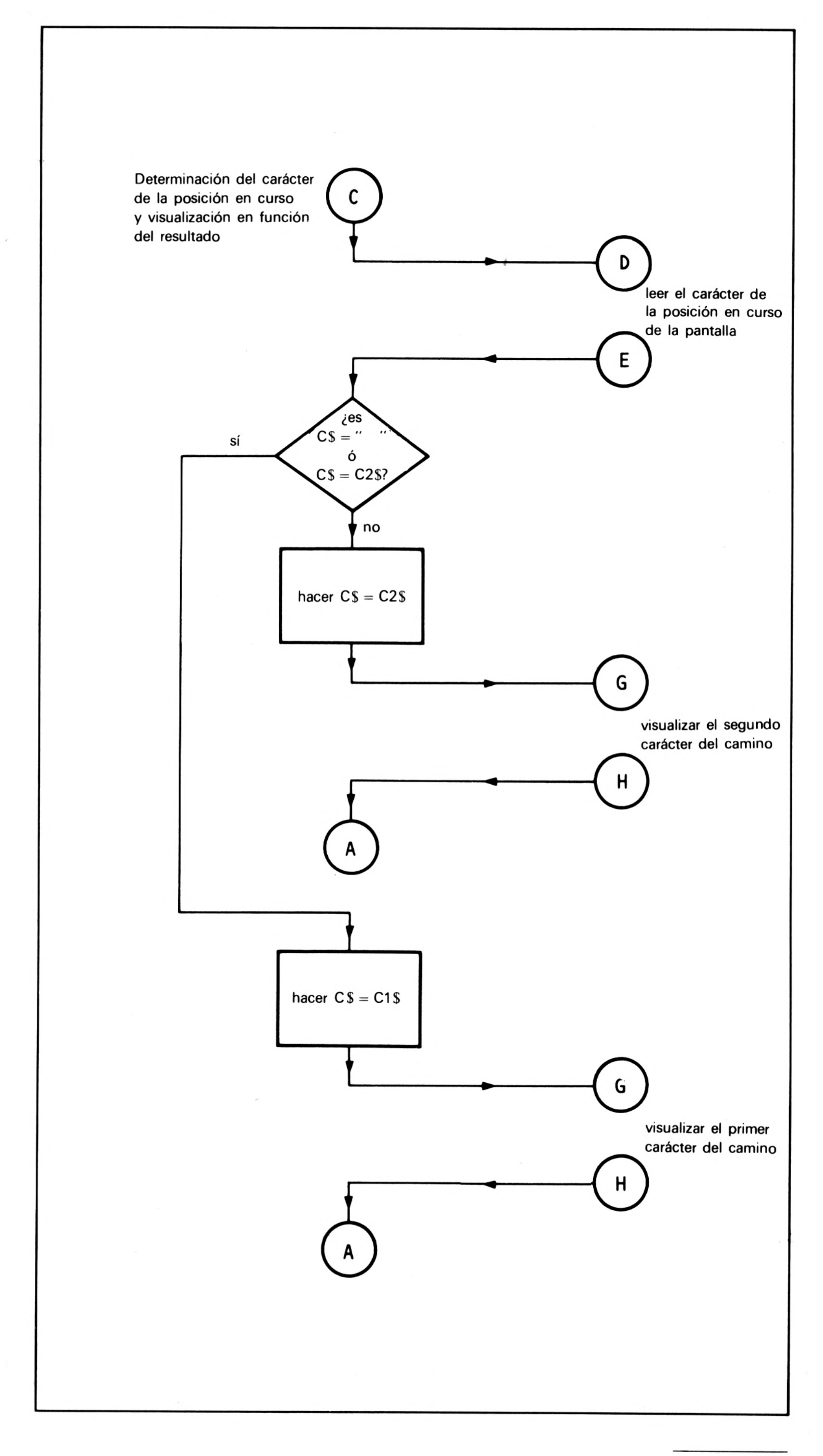


Figura 24.8
Diagrama de flujo del bucle principal de control del programa ejemplo 24.2


Actualización de las coordenadas
de la posición en curso
B
¿es D = 0?
sí
no
¿es D = 1?
sí
no
¿es D = 2?
sí
no
ir a la izquierda
X = X − 1
ir hacia abajo
Y = Y − 1
ir a la derecha
X = X + 1
ir hacia arriba
Y = Y + 1
C


Determinación del carácter
de la posición en curso
y visualización en función
del resultado
C
D
leer el carácter de
la posición en curso
de la pantalla
E
¿es
C$ = " "
ó
C$ = C2$?
sí
no
hacer C$ = C2$
G
visualizar el segundo
carácter del camino
H
A
hacer C$ = C1$
G
visualizar el primer
carácter del camino
H
A

```
1320 IF P >= 0 AND P <= 1 THEN 1335
1325 PRINT "INTRODUCE UN NUMERO COMPRENDIDO ENTRE
           0 Y 1"
1330 GOTO 1315
1335 PRINT "INTRODUCE DOS CARACTERES PARA EL
           CAMINO"
1340 INPUT C1$, C2$
1345 PRINT "INTRODUCE UN NUMERO POSITIVO PARA
           INICIALIZAR"
1350 PRINT "EL GENERADOR DE NUMEROS ALEATORIOS"
1355 INPUT Z
1360 LET Z = RND(-ABS(Z))
1365 REM
1400 REM LIMPIEZA DE PANTALLA
1405 GOSUB 9000
1410 REM
1500 REM VISUALIZACION DEL PRIMER CARACTER
1505 LET X = INT(A/2)
1510 LET Y = INT(H/2)
1515 LET C$ = C1$
1520 GOSUB 9100: REM VISUALIZACION DEL CARACTER
1525 LET D = 0
```

Bucle principal de control

```
1600 REM BUCLE PRINCIPAL DE CONTROL
1605 LET R = RND(1)
1610 IF R > P THEN 1900
1615 IF R > P/2 THEN 1800
1620 REM
1700 REM GIRO A LA DERECHA
1705 LET D = D + 1 - INT ((D + 1)/4) * 4
1710 GOTO 1900
1715 REM
1800 REM GIRO A LA IZQUIERDA
1805 LET D = D - 1 + INT((4 -D)/4) * 4
1810 REM
1900 REM ACTUALIZACION DE LAS COORDENADAS DE LA
         POSICION ACTUAL
1905 IF D = 0 THEN 1950
1910 IF D = 1 THEN 1940
1915 IF D = 2 THEN 1930
1920 LET X = X - 1: REM D = 3
1925 GOTO 2000
1930 LET Y = Y - 1: REM D = 2
1935 GOTO 2000
1940 LET X = X + 1
1945 GOTO 2000
1950 LET Y = Y + 1
1955 REM
2000 REM DETERMINACION DEL CARACTER EN LA POSI
         CION ACTUAL, Y
2005 REM VISUALIZACION DEL CARACTER EN FUNCION
         DEL RESULTADO
```

```
2010 GOSUB 9200: REM LECTURA DEL CARACTER DE LA
                     PANTALLA
2015 IF C$ = " " OR C$ = C2$ THEN 2035
2020 LET C$ = C2$
2025 GOSUB 9100: REM VISUALIZACION DEL SEGUNDO
                     CARACTER DEL CAMINO
2030 GOTO 1600 : REM CONTINUAR BUCLE
2035 LET C$ = C1$
2040 GOSUB 9100 : REM VISUALIZACION DEL PRIMER
                      CARACTER DEL CAMINO
2045 GOTO 1600 : REM CONTINUAR BUCLE
2050 REM
9000 REM INCLUYE SUBPROGRAMAS DE LECTURA Y ESCRI
         TURA DEL CARACTER
9100 REM Y LIMPIEZA DE LA PANTALLA
9200 REM
9999 END
```

Puntos de interés

- El programa se acortaría considerablemente recurriendo a las construcciones **IF ... THEN** e **IF ... THEN ... ELSE**.
- El programa continúa indefinidamente, hasta que el usuario decide interrumpirlo.
- Si el avance de la trayectoria es tan rápido que no se la distingue, puede añadirse un módulo que reduzca la velocidad de funcionamiento.

24.9 Gráficos interactivos

Las técnicas gráficas y los programas expuestos en este capítulo permiten generar en la pantalla imágenes fijas y móviles. Los **gráficos interactivos** dan al usuario la oportunidad de intervenir en la generación de un gráfico a través del teclado.

Estos gráficos abren numerosas posibilidades, como juegos de tablero, "dibujo" directo en la pantalla y corrección de figuras generadas por un programa. Por desgracia, no son muchas las versiones del Basic con posibilidad de realización de gráficos interactivos. En los casos en que se disponga de tal opción, es recomendable escribir subprogramas que ejecuten las siguientes operaciones elementales:

- Llevar el cursor a una posición dada de la pantalla e invitar al usuario a utilizar el teclado; como resultado, aparecen en pantalla el carácter o los caracteres tecleados.

- Permitir al usuario llevar el cursor a cualquier punto de la pantalla y utilizar el teclado; como resultado, aparecen en pantalla los caracteres introducidos y las coordenadas que marcan la posición del primero de ellos.

24.10
Conclusión

En este capítulo hemos discutido los principios generales de la creación de gráficos mediante ordenador y algunas técnicas gráficas elementales. Además, hemos escrito subprogramas para ejecutar esas técnicas en una versión concreta del Basic, pero provistos de un interfaz de conexión con el programa principal independiente del ordenador. Los dos programas ejemplo examinados utilizan los subprogramas que acaban de citarse.

La creación de gráficos con ordenador es una actividad compleja que no deja de crecer. Todo lo que puede hacerse en este capítulo es presentar algunas ideas generales, ilustrar unas pocas técnicas y recordar al lector que una obra de arte —sea una pintura o sea un programa— es algo más que la mera aplicación de una técnica.

24
Ejercicio

1. ¿Qué se entiende por resolución al hablar de gráficos creados con ordenador?
2. Vuelva a escribir los subprogramas de borrado de pantalla, escritura y lectura de caracteres en la versión del Basic que soporte su ordenador. Asegúrese de que el interfaz que lo une al programa encargado de llamar permanece invariable.
3. A continuación daremos las especificaciones de una serie de subprogramas adicionales de "pinceladas en Basic", que pueden escribirse en una versión de ejecución práctica o en la de referencia utilizando el subprograma de escritura de caracteres.
   a) Dibujar en pantalla el contorno de un rectángulo. Los parámetros transferidos al subprograma son las coordenadas de los vértices inferior izquierdo y superior derecho y el carácter encargado de trazar el perímetro. No hay que borrar ni sombrear el área interior de la figura.
   b) Sombrear el interior de un rectángulo con una densidad uniforme. Los parámetros transferidos al subprograma son las coordenadas de los vértices inferior izquierdo y superior derecho, la densidad de sombreado (una fracción comprendida entre 0 y 1) y el carácter utilizado para ello. El sombreado se realiza mediante la misma técnica de números aleatorios descrita en el programa ejemplo 24.1.

No borre ni escriba áreas de caracteres, porque así el subprograma puede usarse acumulativamente.

c) Las fórmulas que dan las coordenadas X e Y de los puntos de una circunferencia son:

Arco superior: $Y = B + \sqrt{R^2 - (X - A)^2}$

Arco inferior: $Y = B - \sqrt{R^2 - (X - A)^2}$

siendo A y B las coordenadas del centro de la circunferencia y R el radio; los valores de X están comprendidos entre $A - R$ y $A + R$.

Salvo que los espacios para caracteres de la pantalla sean cuadrados, la fórmula generará en realidad una elipse, pero puede modificarse para compensar la forma de dichos espacios. La fórmula puede utilizarse en un subprograma para generar una circunferencia (o una elipse) o para sombrear el interior de zonas circulares (o elípticas). Los parámetros son las coordenadas del centro, el radio, el carácter utilizado para marcar el contorno o para sombrear y, en este último caso, la densidad de sombreado.

d) La fórmula que da la línea recta comprendida entre dos puntos de coordenadas (P, Q) y (R, S) es:

$$Y = Q + \frac{(X - P)(S - Q)}{R - P}$$

siendo (X, Y) las coordenadas de un punto cualquiera de la recta y estando X comprendido entre P y R.

Utilice esta fórmula como base de un subprograma capaz de trazar una recta entre dos puntos cualesquiera. Los parámetros del programa son las coordenadas de los puntos y el carácter utilizado para dibujar la recta.

Nota: Al escribir los subprogramas descritos, hay que decidir entre dejar que las figuras resultantes crucen los límites de la pantalla o hacer que, en caso de cruce, se produzca una situación de error y no se ejecute el gráfico.

4. Vuelva a escribir uno de los programas ejemplo 24.1 y 24.2, o los dos, utilizando la construcción **IF ... THEN ... ELSE** y agrupando varias instrucciones en una misma línea.

5. Escriba un subprograma de tipo general que trace histogramas dibujando las bandas verticales a una velocidad adecuada. Los parámetros podrían ser los siguientes:

la escala vertical,
un nombre para el eje vertical,
el número de bandas que han de trazarse y la altura de cada una,
un nombre para el eje horizontal,
un nombre para cada banda,
un título para el gráfico completo.

Diseñe un programa principal adecuado para utilizar el subprograma o incorpórelo a alguno que haga uso de histogramas.

6. Llame la atención hacia una zona rectangular de la pantalla haciendo que los caracteres contenidos en la misma emitan destellos de forma intermitente. Para lograrlo se leen los elementos de esa zona y se transfieren a una matriz para volverlos de nuevo a la pantalla y así sucesivamente. Los parámetros son las coordenadas de los vértices

inferior izquierdo y superior derecho del rectángulo y la frecuencia y duración de los destellos.

7. Utilizando símbolos gráficos adecuados, diseñe un grupo de caracteres alfabéticos o numéricos (o de las dos clases) que ocupen, por ejemplo, cinco por nueve espacios de pantalla. Almacene los símbolos adecuadamente ordenados en los elementos de una matriz, provistos de un índice que identifique el carácter.

Escriba a continuación un subprograma que presente en una posición dada de la pantalla uno de esos grandes caracteres. La posición puede especificarse, por ejemplo, mediante las coordenadas del vértice inferior izquierdo del carácter. Este se identifica por el valor del índice en la matriz.

8. Escriba un programa gráfico de tipo general que acepte como información para crear figuras una serie de coordenadas y los caracteres que deben escribirse en cada una de ellas. La formación de la imagen en pantalla puede hacerse al término de la introducción de todas las coordenadas y de los caracteres o en tiempo real, conforme dicha introducción vaya produciéndose.

9. Sugerencias para programas gráficos: Las técnicas descritas en este capítulo y en las preguntas anteriores de este ejercicio permiten escribir muchísimos programas de creación de imágenes, y a continuación damos algunas sugerencias. En todos los casos se recomienda especificar rigurosamente el programa antes de diseñarlo y trazar primero los gráficos en papel cuadriculado; todas las funciones dependientes del ordenador deben escribirse en forma de subprogramas independientes.

- Un programa que dibuje mapas sencillos para, por ejemplo, ilustrar conceptos como el de perímetro.
- Un juego de aritmética que utilice cifras de gran tamaño para indicar las operaciones a realizar.
- Un juego de barcos.
- Un programa capaz de dibujar proyecciones en un plano de cuerpos tridimensionales sencillos y someterlas a un movimiento de rotación.
- Si su ordenador permite la creación de gráficos interactivos, haga un programa para jugar a tres en raya.
- Escriba programas para dibujar espirales, estrellas, medias lunas y otras formas similares.
- Escriba programas para crear dibujos animados sencillos.
- Escriba programas adecuados para ejecutar una diversidad grande de curvas.


RESET
RETURN

25 Análisis de trayectorias críticas

Este capítulo expone una técnica de dirección empresarial conocida como **análisis de trayectorias críticas.** Expondremos algunos conceptos asociados a dicha técnica y estudiaremos un programa capaz de llevarlos a la práctica.

Es esta una aplicación informática de gran interés, que sirve además para mejorar el dominio del diseño de programas. De todas formas, este material no volverá a tratarse y, por tanto, quien lo desee puede saltárselo sin temor a romper la continuidad del texto.

25.1 El contexto del análisis de trayectorias críticas

El análisis de trayectorias críticas es propio de la organización de proyectos de gran envergadura, como la construcción de un puente, la producción de una obra de teatro o el seguimiento de una ley en las cámaras legislativas. Todas estas empresas constan de varias operaciones independientes que deben ser ejecutadas por equipos de personas también diferentes. El objetivo del análisis de trayectorias críticas es determinar la secuencia o las secuencias de operaciones que

son críticas para la terminación de toda la obra dentro del plazo de tiempo prefijado.

25.2

Actividades y sucesos

El análisis de trayectorias críticas se describe mediante unos pocos términos técnicos con los que es preciso familiarizarse antes de seguir más adelante. Cada una de las operaciones independientes de que consta una obra se llama **actividad.** Una actividad tiene un principio y un final, y su realización exige cierto tiempo o **duración.** Se supone que antes de iniciar un trabajo se conocen las duraciones de todas las actividades que lo integran. Una actividad puede ilustrarse como una línea que une su principio con su final (véase la figura 25.1).

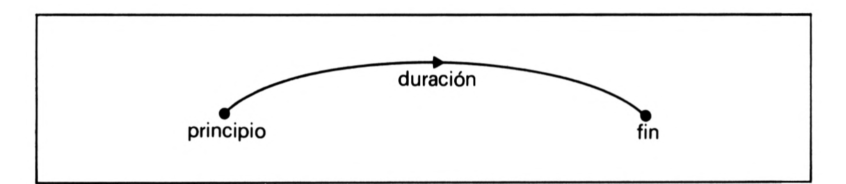


Figura 25.1
Una actividad

Las relaciones que se traban entre las diversas actividades son muy importantes, porque algunas son independientes, pero las hay que sólo pueden empezar cuando otras han terminado. Para representar esta situación basta unir los puntos de principio y final de las actividades. El punto en que confluyen varias se llama **suceso.** Cada suceso puede considerarse como una etapa cubierta en el desarrollo del trabajo y, como ilustra la figura 25.2, puede representarse mediante un punto situado en la confluencia de las líneas de actividad. En esa misma figura, la actividad que empieza en el suceso sólo puede iniciarse cuando han concluido todas las que confluyen en aquél.

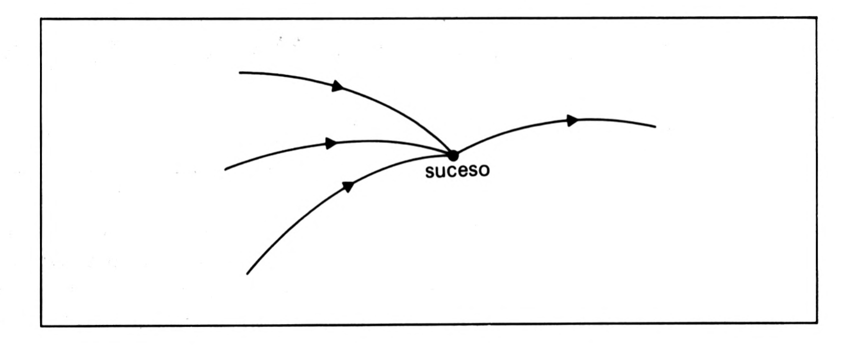


Figura 25.2
Un suceso

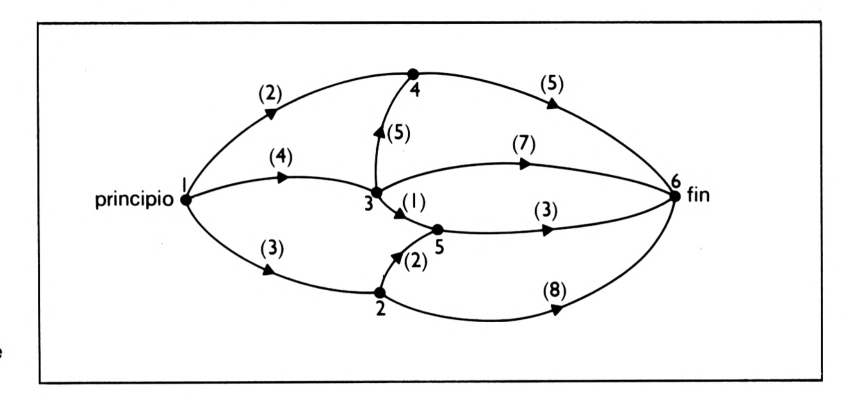


Figura 25.3
Red de actividades y sucesos de un proyecto

25.3 Redes

El resultado de dibujar todas las líneas correspondientes a las actividades de una empresa y de conectar los diversos sucesos es una **red** de actividades y sucesos. La figura 25.3 ilustra la red correspondiente a un trabajo sencillo, y merece algunos comentarios de interés:

- Los sucesos se numeran ordenadamente desde el origen del trabajo, de manera que ninguna actividad conduce de uno mayor a otro menor.
- La duración de cada actividad se escribe entre paréntesis.
- Cualquier actividad puede describirse por los sucesos que conecta. Así, la que conecta los sucesos 1 y 3 tiene una duración 4.

Se llama **trayectoria** a cualquier secuencia de actividades que conecte el principio y el final del trabajo. En la figura a que nos estamos refiriendo, un ejemplo de trayectoria sería la que conecta el suceso 1 con el 3 y éste con el 6.

25.4 Fechas primera y última

Una forma de encarar la realización de un proyecto es empezar todas las actividades lo antes posible, es decir: en cuanto todas las que conducen a un suceso dado terminan, comienzan todas las que parten de ese mismo suceso. Esta fecha de inicio es la **fecha primera** de un suceso determinado, que marca el instante mismo en que terminan todas las actividades que confluyen en un suceso.

Pero si todas las operaciones comienzan lo antes posible, algunas de las que conducen al suceso final del proyecto acabarán demasiado pronto. Esto sugiere otra forma de llevar el trabajo, a saber: el momento de inicio de cualquier actividad se retrasa lo más posible, pero de forma que todas las que conduzcan al final del proyecto terminen a la vez. En este caso, la fecha de inicio se llama **fecha última** de un suceso determinado y marca la última oportunidad en cuanto a tiempo que tiene cada actividad para empezar de forma tal que la obra concluya en el momento previsto.

25.5

La trayectoria crítica

Si se calculan las fechas primera y última de todos los sucesos de una red, se hallará al menos una trayectoria en la que ambas fechas serán iguales para cada uno de los sucesos que recorra: esa es la **trayectoria crítica.** La duración de todas las actividades de una trayectoria crítica es igual a la duración total del proyecto, y cualquier retraso en una de ellas provocará un retraso en el proyecto completo.

25.6

Cálculo de las fechas primera y última

El objetivo del análisis de trayectorias críticas es determinar las fechas primera y última de todos los sucesos de un proyecto y, de esa forma, determinar la trayectoria o trayectorias críticas. Expondremos a continuación una forma de calcular las fechas primeras y diseñaremos un programa que lo lleve a la práctica. El cálculo de las fechas últimas se tratará en el ejercicio propuesto al final del capítulo.

Un algoritmo informal para calcular fechas primeras sería el siguiente:

Igualar a cero la fecha primera del suceso de partida.
Recorrer los restantes sucesos de la red por orden desde el principio de la misma.
Al llegar a cada suceso:

- localizar todas las actividades que terminan en él;
- para cada actividad, sumar su duración a la fecha primera del suceso de partida;
- el máximo de los valores así obtenido será la fecha primera del suceso en cuestión.

Consideremos, por ejemplo, la porción de red ilustrada en la figura 25.4. Las fechas primeras ya calculadas figuran entre corchetes. Al aplicar a las tres actividades que confluyen en el suceso 6 el método que acaba de exponerse, resulta:

Actividad	*Fecha primera del suceso de partida*	*Duración*	*Fecha primera + duración*
3 a 6	6	5	11
4 a 6	8	4	12
5 a 6	7	2	9
			Valor máximo: 12

Por tanto, la fecha primera del suceso 6 es 12.

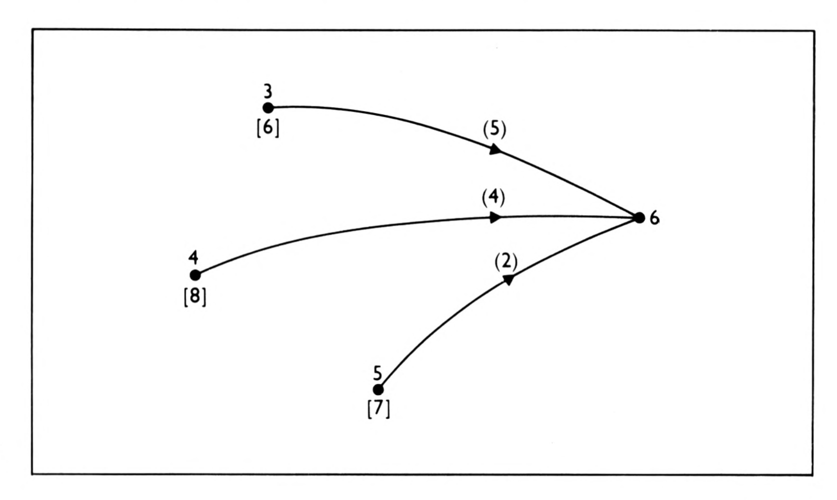


Figura 25.4
Parte de una red de actividades y sucesos

25.7
Programa ejemplo 25.1

Una primera descomposición del programa encargado de calcular fechas primeras sería:

Introducir los sucesos inicial y final y la duración de cada actividad.
Calcular la fecha primera de cada uno de los sucesos.
Mostrar la fecha primera de cada uno de los sucesos.

El siguiente paso sería decidir las variables y las estructuras de datos que han de utilizarse. Las estructuras se eligen teniendo en cuenta que el programa va a ampliarse para incluir el cálculo de fechas últimas. Una posibilidad, pero no la única, desde luego, es

utilizar matrices bidimensionales, una para las actividades y otra para los sucesos.

La matriz de actividades contiene una columna con sucesos de partida, otra con sucesos finales y una tercera con duraciones, ordenadas de forma que cada fila hace referencia a una misma actividad. La matriz de las actividades correspondientes a la figura 25.3 se recoge en la tabla de abajo. Si se emplea la variable **A** para denominarla, el elemento **A(4, 3)** es el que se encuentra en la confluencia de la tercera columna con la cuarta fila, que corresponde a la duración de la actividad que va del suceso 2 al 5.

Suceso inicial	*Suceso final*	*Duración*
1	2	3
1	3	4
1	4	2
2	5	2
2	6	8
3	4	5
3	5	1
3	6	7
4	6	5
5	6	3

Matriz de actividades de la red de la figura 25.3

La matriz de sucesos tiene una columna para fechas primeras, otra para fechas últimas y una tercera para las diferencias o **márgenes.** Cada una de las filas corresponde a un solo suceso, y además el número de la misma corresponde al número del suceso. Así, si esta matriz se identifica mediante la variable **E**, el elemento **E(5, 1)** corresponde a la fecha primera del suceso 5.

Como es habitual, es preciso declarar el límite superior del tamaño de la matriz, lo que impone un tope al número de actividades y sucesos de la red procesables por el programa. En este caso utilizaremos un límite de 25.

Definidas las estructuras de datos, procederemos a un nuevo refinamiento del programa:

Repetir la secuencia;

introducir el número de actividades y el número de sucesos

hasta quedar por debajo del límite de la matriz.

Introducir los elementos de la matriz de actividades: sucesos de partida, sucesos finales y duraciones.

Igualar a cero la fecha primera del suceso de partida.

Para cada uno de los demás sucesos de la matriz:

igualar la fecha primera a cero;
localizar todas las actividades que terminan en un suceso;
para cada una de dichas actividades:

sumar a la duración la fecha primera del suceso de partida;
si este valor es mayor que la fecha primera del suceso en cuestión, sustituir por él dicha fecha.

Observe de qué forma se determina el máximo: se elige un valor inicial cero y, si cualquiera de los valores posteriores es mayor que el valor en curso, lo sustituye. El programa ya está especificado con detalle suficiente y puede escribirse.

Variables

A(25, 3)	Matriz de actividades.
E(25, 3)	Matriz de sucesos.
N1	Número de actividades.
N2	Número de sucesos.
K, L	Contadores de bucle.
T	Variable provisional.

Diagrama de flujo

La figura 25.5 ilustra el flujo de control del programa ejemplo 25.1.

Programa

```
100 REM PROGRAMA EJEMPLO 25.1
105 REM ANALISIS DE TRAYECTORIAS CRITICAS
110 REM CALCULO DE LAS PRIMERAS FECHAS SOLAMENTE
115 DIM A(25, 3), E(25, 3)
120 REM
200 REM SEGMENTO DE INTRODUCCION
205 PRINT "NUMERO DE ACTIVIDADES?";
210 INPUT N1
215 PRINT "NUMERO DE SUCESOS?";
220 INPUT N2
225 IF (N1 <= 25) AND (N2 <= 25) THEN 250
230 PRINT "NO PUEDE HABER MAS DE 25 NUMEROS"
235 PRINT "POR FAVOR SIMPLIFIQUE SU RED"
240 PRINT "Y VUELVA A INTRODUCIRLOS"
245 GOTO 200
```

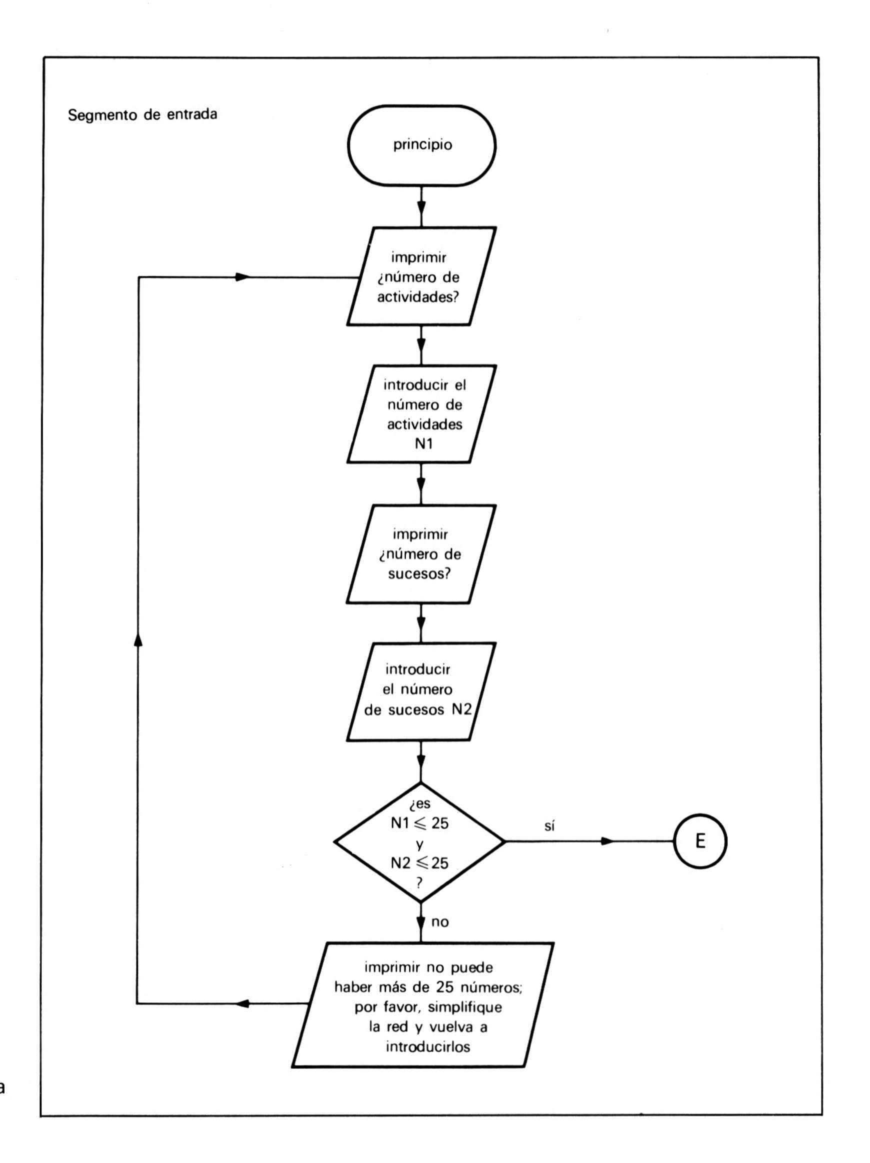
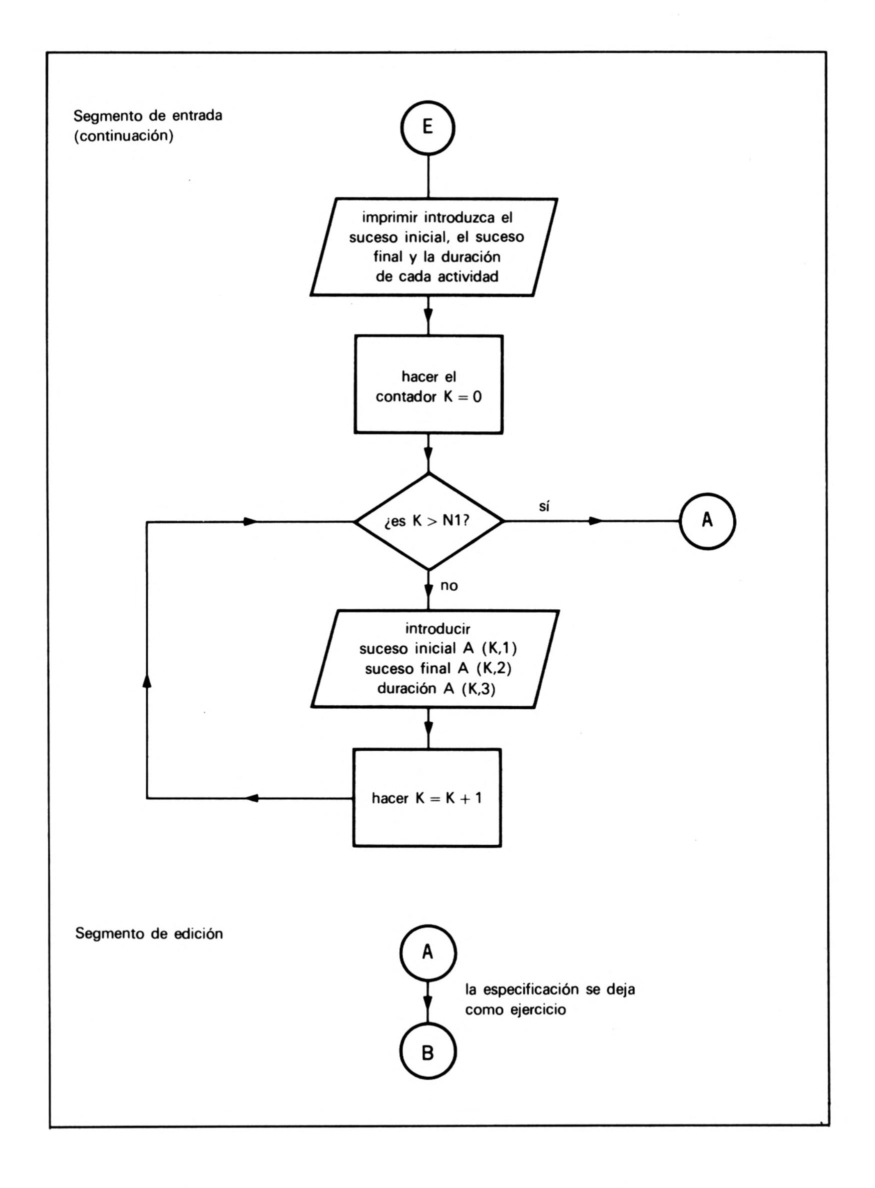
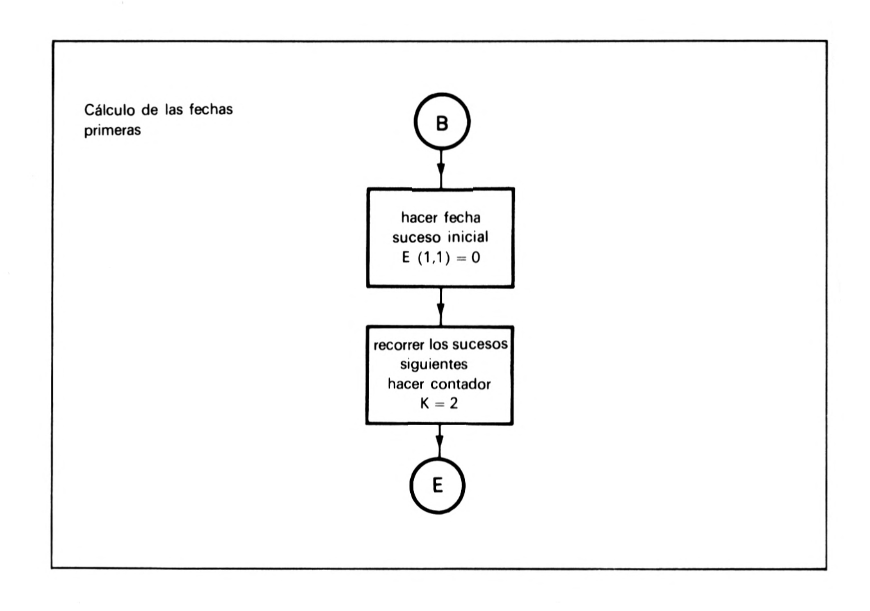


Figura 25.5
Diagrama de flujo del programa ejemplo 25.1

```
250 PRINT "INTRODUZCA LOS SUCESOS INICIAL Y
          FINAL"
255 PRINT "Y LA DURACION DE CADA ACTIVIDAD"
260 FOR K = 1 TO N1
265 INPUT A(K, 1), A(K,2), A(K, 3)
270 NEXT K
275 REM
300 REM ESPACIO PARA EL SEGMENTO DE EDICION
305 REM
400 REM CALCULO DE LAS PRIMERAS FECHAS
405 LET E(1, 1) = 0: REM SUCESO INICIAL
```



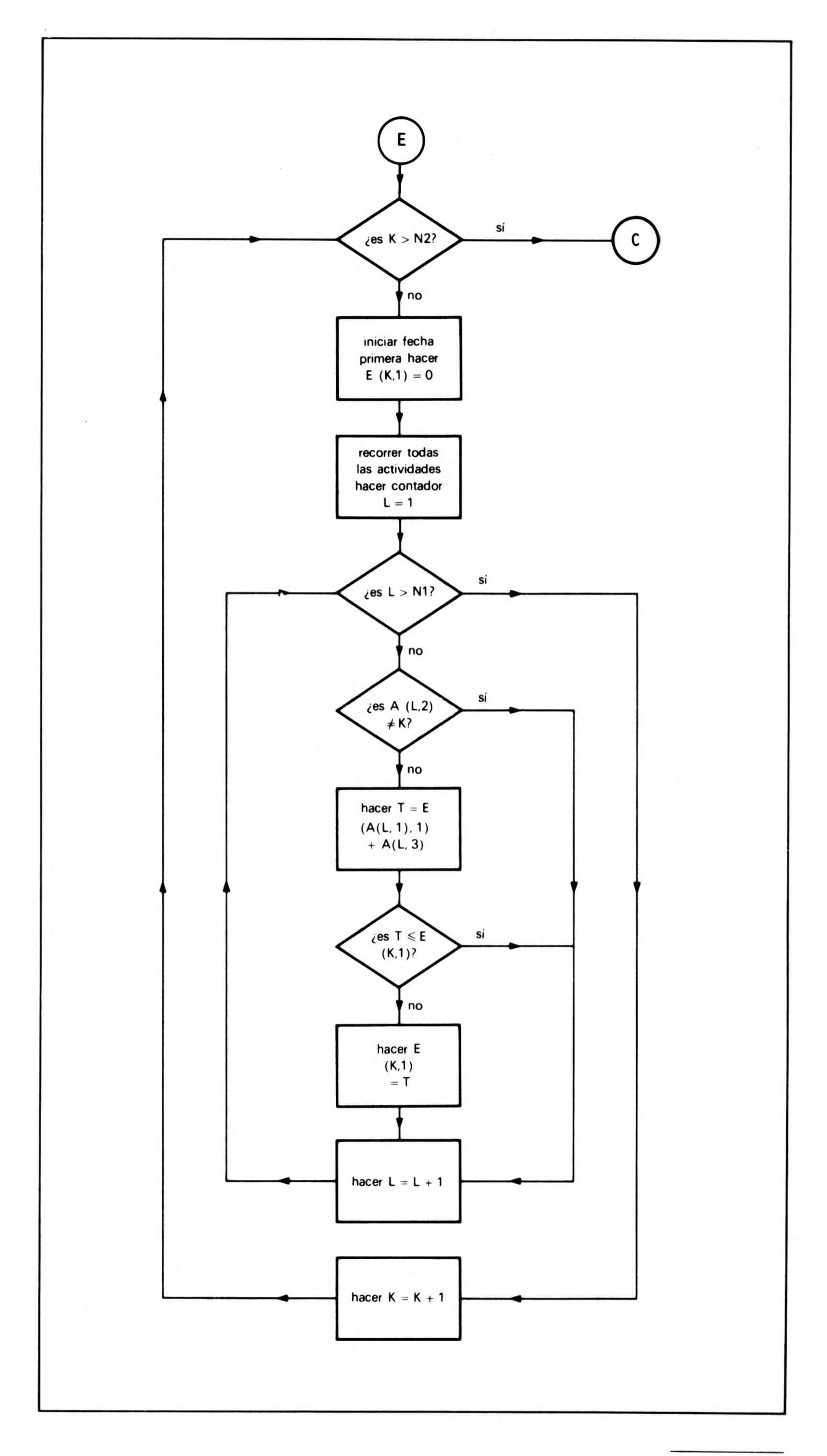

```
410 FOR K = 2 TO N2: REM RECORRER LOS
                         SUCESOS SIGUIENTES
415 LET E(K, 1) = 0: REM INICIALIZACION DE LA
                         PRIMERA FECHA
420 FOR L = 1 TO N1: REM RECORRER TODAS LAS ACTI
                         VIDADES
425 REM RECHAZAR TODAS LAS ACTIVIDADES QUE NO
        FINALICEN EN EL SUCESO K
430 IF A(L, 2) <> K THEN 465
435 REM CALCULO DE LA PRIMERA FECHA DEL SUCESO
440 REM INICIAL, MAS SU DURACION
```



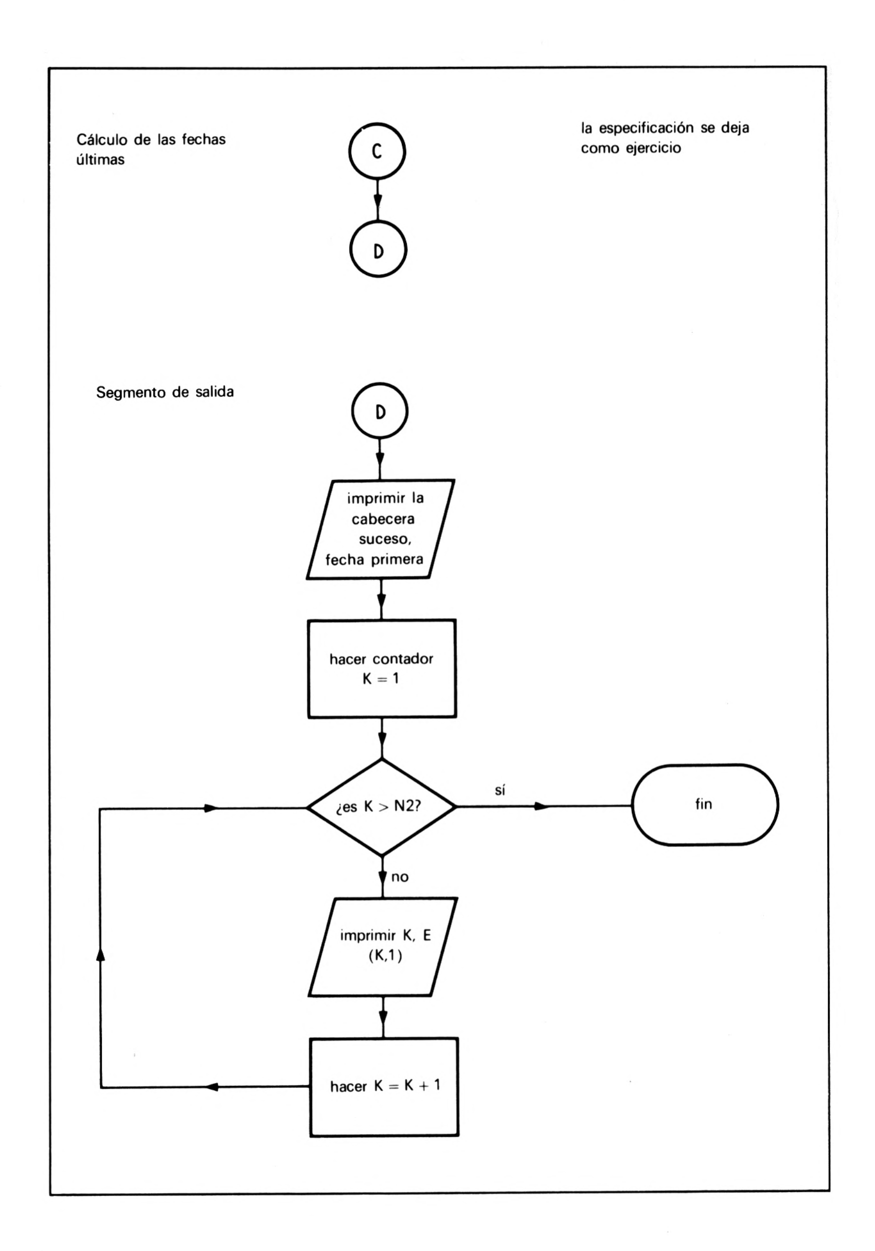

```
445 LET T = E(A(L, 1), 1) + A(L, 3)
450 REM COMPARAR CON LA FECHA PRIMERA ACTUAL
455 IF T <= E(K, 1) THEN 465
460 LET E(K, 1) = T: REM ACTUALIZAR LA PRIMERA
                          FECHA
465 NEXT L
470 NEXT K
475 REM
500 REM ESPACIO PARA EL CALCULO DE LA ULTIMA
        FECHA
505 REM
600 REM SEGMENTO DE SALIDA
605 PRINT
610 PRINT "SUCESO"; TAB(10); "FECHA PRIMERA"
615 FOR K = 1 TO N2
620 PRINT K; TAB(10); E(K, 1)
625 NEXT K
630 REM
700 END
```

Puntos de interés

- En la línea 445, el valor de un elemento de la matriz **A** actúa como índice de otro elemento de la **E**.
- Se ha previsto espacio para un segmento de edición y para el cálculo de fechas finales.
- La condición de la línea 455 es la de no actualizar el máximo: si el valor de **T** es menor o igual al valor en curso, no se emprende ninguna acción.



E
¿es K > N2?
sí
C
no
iniciar fecha
primera hacer
E (K,1) = 0
recorrer todas
las actividades
hacer contador
L = 1
¿es L > N1?
sí
no
¿es A (L,2)
≠ K?
sí
no
hacer T = E
(A(L, 1), 1)
+ A(L, 3)
¿es T ≤ E
(K,1)?
sí
no
hacer E
(K,1)
= T
hacer L = L + 1
hacer K = K + 1



25.8
Conclusión

Hemos estudiado en este capítulo la técnica de análisis de trayectorias críticas y algunos conceptos asociados a ella, como son los de actividad, suceso, red, fecha primera, fecha última y margen.

El análisis de trayectorias críticas es un instrumento muy útil para la dirección de proyectos. En grandes empresas casi nunca se utiliza sola, sino incluida en un contexto más amplio llamado PERT (técnica de evaluación y revisión de proyectos). Sin la ayuda de tales instrumentos sería casi imposible organizar con eficacia los proyectos de gran envergadura.

El ejercicio que sigue le permitirá completar el programa diseñado en este capítulo, así como ampliarlo y aplicarlo a varias situaciones diferentes.

25

Ejercicio

1. Defina en pocas palabras los siguientes términos: actividad, suceso, fechas primera y última, margen, trayectoria crítica.
2. Si una actividad no se encuentra en el camino crítico, ¿podrá retrasarse indefinidamente sin comprometer la terminación del trabajo? Explique la respuesta en términos de los conceptos definidos en la pregunta anterior.
3. Calcule manualmente las fechas primeras de los sucesos de la figura 25.3 y utilícelas para probar el programa del ejemplo. Compare los resultados obtenidos por uno y otro procedimientos.
4. Un algoritmo informal para el cálculo de fechas últimas podría ser el siguiente:

   Igualar la fecha última del suceso final de la red a su fecha primera.
   Recorrer hacia atrás todos los demás sucesos de la red.
   Para cada uno de ellos:

   - localizar todas las actividades que parten de él;
   - para cada una de las actividades restar la duración a la fecha última del suceso final;
   - el mínimo de los valores así obtenidos es la fecha última del suceso en cuestión.

   a) Aplique este método a la porción de red representada en la figura 25.6. Las fechas últimas ya calculadas figuran entre corchetes.
   b) Diseñe un segmento de programa capaz de realizar el anterior algoritmo. Utilice las estructuras de datos y las variables ya definidas. El segmento de programa debe empezar en la línea 500.
   c) Modifique el segmento de salida del programa ejemplo para incluir en él las fechas últimas.
   d) Ejecute el programa modificado con los datos de la figura 25.3 y compare los resultados obtenidos así con los calculados manualmente.
5. Amplíe el programa del ejemplo para permitir al usuario editar datos ya introducidos. Especifique con claridad lo que pretende conseguir con el segmento de edición y diséñelo a continuación. Utilice las líneas 300 a 395.

   El programa puede también ampliarse de forma que pueda llevar a cabo el análisis varias veces seguidas con datos que pueden resultar modificados tras cada ejecución. Utilice este programa modificado para

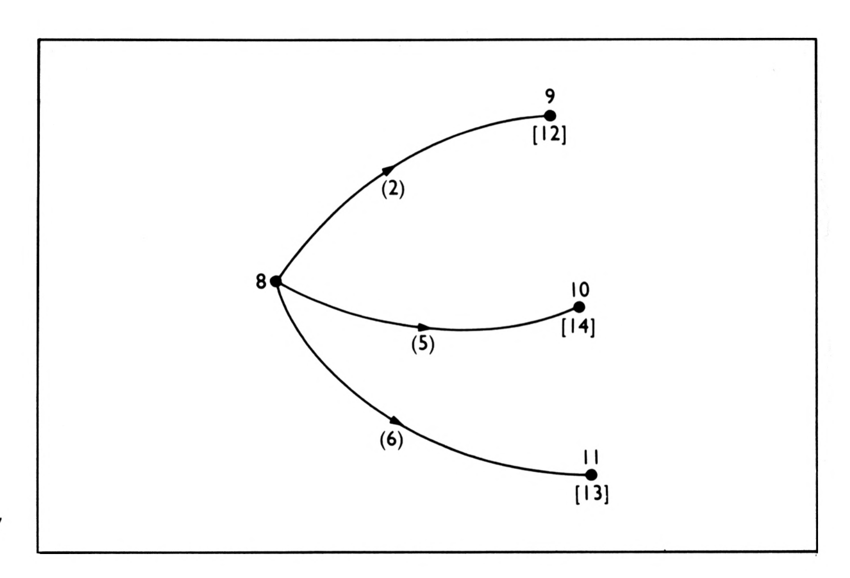


Figura 25.6
Parte de una red de actividades y sucesos

investigar los efectos de las variaciones en la duración de las actividades de la figura 25.3. Estudie tanto las que se encuentran en la trayectoria crítica como las situadas fuera de ella y comente los resultados obtenidos.

6. La construcción de un buque a motor comprende las actividades recogidas en la lista siguiente, ilustradas además en la red de la figura 25.7:

Actividad	*Suceso inicial*	*Suceso final*	*Duración (días)*
Moldeo del casco	1	2	4
Moldeo de la cubierta	1	3	4
Construcción de los mamparos interiores	1	5	6
Unión de la cubierta al casco	3	4	1
Montaje de puertas y escotillas	4	6	6
Montaje de accesorios	1	6	6
Instalación eléctrica y de fontanería	6	7	8
Instalación de los mamparos interiores	5	6	4
Instalación del motor y conexión de sus mandos	6	9	3
Instalación de los accesorios	6	8	10
Pintura del exterior	4	9	3
Pintura del interior	8	9	4

Observe que para no duplicar la numeración de las actividades se han introducido tres **actividades ficticias** que unen los sucesos 2 y 3, 4 y 5 y 7 y 8 y tienen una duración nula. Los datos correspondientes a tales actividades deben introducirse también en el programa.

a) Utilice el programa ya escrito para calcular las fechas primeras y últimas de los sucesos de la red.

b) Investigue las consecuencias de los retrasos sufridos por actividades situadas dentro y fuera de la trayectoria crítica y el resultado de acelerar algunas de tales actividades.

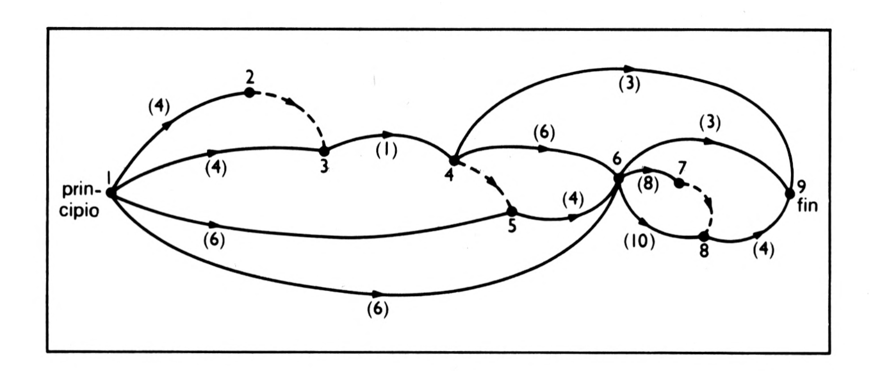


Figura 25.7
Construcción de un buque

7. Diseñe varias redes de actividades y sucesos y analícelas sirviéndose del programa. También puede obtener información de algunos proyectos reales y utilizarla como entrada de dicho programa.


RESET
RETURN

26 Análisis numérico

En este capítulo aplicaremos las técnicas de programación al **análisis numérico,** una rama de las matemáticas que despierta un interés creciente, sobre todo cuando se asocia a la informática. El análisis numérico tiene aplicaciones tan variadas como la determinación de las trayectorias de vuelo de aeronaves, la predicción de tendencias económicas o la previsión del tiempo.

Este capítulo y el que le sigue son los dos únicos del libro que presuponen algunos conocimientos matemáticos en el lector. Aunque los términos y conceptos utilizados se definen brevemente en el momento de su presentación, para sacar partido a esta parte del texto es imprescindible tener una base previa de cálculo y análisis numérico.

Utilizaremos aquí casi todas las técnicas y recursos expuestos en los capítulos precedentes, aunque no se emplearán estructuras de datos complejas. Por su parte, este capítulo guarda una relación estrecha con el siguiente, dedicado a simulación, pero no con los posteriores. Por tanto, los dos pueden omitirse sin pérdida de continuidad.

26.1 Técnicas de análisis numérico

El análisis numérico se refiere a la solución de ecuaciones y a la realización de operaciones como la integración y la diferenciación por

métodos de aproximación basados en secuencias de cálculos relativamente sencillos. Aunque se trata de métodos aproximados, el grado de aproximación siempre está controlado y por lo general pueden obtenerse resultados de exactitud aceptable en un período de tiempo razonable.

Muchas técnicas de análisis numérico consisten en una serie de cálculos que "convergen" hacia un resultado: son las llamadas **técnicas iterativas.** En cada paso, o iteración, se realiza el mismo cálculo a partir del resultado del paso anterior y, como resultado, se obtiene un nuevo valor más exacto. A mayor número de repeticiones, mayor exactitud.

En este capítulo expondremos varias técnicas de análisis numérico. El núcleo del texto lo constituyen un método iterativo de resolución de ecuaciones conocido como **método de trasposición,** el **método de Gauss** de resolución de sistemas de ecuaciones y una técnica de integración conocida como **regla de los trapecios.** En el ejercicio propuesto al final encontrará el lector una exposición del **método de Newton** de resolución de ecuaciones, una aplicación del método de Gauss a la **inversión de matrices** y la **regla de Simpson** de integración numérica.

26.2

Soluciones aproximadas de ecuaciones: método de trasposición

Gran número de ecuaciones matemáticas —las que contienen funciones trigonométricas y exponenciales o las polinómicas de orden superior a 3, por ejemplo— no pueden ser resueltas con toda precisión, y deben resolverse por métodos de iteración numérica.

Veamos a continuación un ejemplo del **método de trasposición,** que puede aplicarse a casos muy variados.

Ejemplo

La ecuación

$$x^4 - x - 1 = 0$$

tiene una solución próxima a $x = 1$, porque la expresión del miembro izquierdo pasa de -1 a 13 cuando x lo hace de 1 a 2. Una forma de trasponer esta ecuación sería:

$$x^4 = x + 1$$

$$x = \sqrt[4]{x + 1}$$

La ecuación se ha traspuesto para que x quede sola a la izquierda. En muchos casos, semejante trasposición tiene la propiedad de que si a la derecha se sustituye x por una estimación de la solución, se obtiene a la izquierda una nueva estimación más exacta. En otras palabras: si x_n es la estimación enésima de la solución, x_{n+1}, que corresponde a la estimación $(n + 1)$ viene dado por:

$$x_{n+1} = \sqrt[4]{x_n + 1}$$

La tabla de abajo recoge algunos valores de x_n obtenidos de esta forma, partiendo de la estimación inicial $x_1 = 1$.

n	x_n	$x_n + 1$	$x_n + 1 = \sqrt[4]{x_n + 1}$
1	1	2	1,1893
2	1,1893	2,1893	1,2165
3	1,2165	2,2165	1,2201
4	1,2201	2,2201	1,2207
5	1,2207	etc.	

Puede verse que tras tan sólo cuatro iteraciones, los valores de x **convergen** ya hacia la solución aproximada $x = 1{,}22$.

Si una trasposición de una ecuación no da lugar a una serie convergente de aproximaciones, se prueba con otra diferente. La diferencial del miembro derecho de la trasposición indica si ésta producirá o no una serie convergente. Basta para ello que la derivada sea de magnitud inferior a 1 en la región de la solución. Es decir, si la ecuación traspuesta es

$$x = g(x)$$

la trasposición generará una serie convergente de aproximaciones si

$$\left|\frac{d}{dx}(g(x))\right| < 1$$

en la región de la solución.

26.3

Programa ejemplo 26.1

El objetivo de este programa es implementar el método de trasposición para calcular soluciones aproximadas de una ecuación que acabamos de exponer. La técnica requiere tres elementos de información:

la ecuación traspuesta, una estimación inicial de la solución y el grado de exactitud deseado, que se fija como diferencia absoluta entre dos estimaciones consecutivas de la solución. En otras palabras, si c es el mencionado grado de exactitud, la iteración se detiene cuando

$$|x_{n+1} - x_n| < c$$

Puede ocurrir que el proceso de iteración no converja hacia la solución, lo que obliga a incluir en el programa instrucciones de salvaguarda para que el proceso no continúe indefinidamente.

Por desgracia, la ecuación traspuesta no puede introducirse como dato, lo que significa que es preciso modificar una línea del programa cada vez que vaya a ensayarse una nueva ecuación o una nueva trasposición de la misma ecuación.

Diseño del programa

Las fases del proceso son:

Introducir la estimación inicial de la solución y el grado de exactitud.

Escribir la ecuación traspuesta en una línea adecuada del programa.

Resolver la ecuación con el grado de exactitud deseado.

Mostrar a la salida.

La forma más sencilla de poder cambiar una línea de programa es escribirla como subprograma y dar al usuario las instrucciones pertinentes para hacer el listado de dicho subprograma y crear esa línea.

La única parte del programa que exige una especificación más detallada es la correspondiente al paso "resolver la ecuación". Esta fase precisa de dos variables, **XANT**, el antiguo valor de x, y **XNUE** el valor de x actualizado. He aquí un algoritmo del proceso de resolución:

Igualar **XNUE** a la estimación inicial de la solución.
Poner a cero el contador de iteración.
Repetir el proceso:

Transferir el valor de **XNUE** a **XANT**.
Llamar al subprograma ecuación para obtener el valor actualizado de **XNUE**.
Incrementar en 1 el contador de iteración.

Hasta que |**XNUE** – **XANT**| < grado de exactitud deseado, o hasta que el contador de iteración exceda de cierto límite.

Si la iteracíón se detiene en la condición primera, significa que se ha encontrado una solución. Si lo hace en la segunda, el proceso ha fallado.

La estructura del programa está ya especificada con detalle suficiente y puede pasarse al proceso de escritura. En este ejemplo utilizaremos la ecuación de la sección anterior.

Variables

X1 Valor antiguo de **X**, parámetro de subprograma.
X2 Valor nuevo de **X**, parámetro de subprograma.
N Contador de iteración.
X0 Estimación inicial de la solución.
C Grado de exactitud necesario.

Diagrama de flujo

Véase la figura 26.1.

Programa

```
100 REM PROGRAMA EJEMPLO 26.1
105 REM TRASPOSICION: METODO APROXIMADO
110 REM PARA HALLAR LA SOLUCION DE ECUACIONES
115 REM
```

Instrucciones para el usuario y entrada

```
200 REM INSTRUCCIONES DEL USUARIO E INTRODUCCION
        DE DATOS
205 PRINT "APROXIMACION A LA SOLUCION DE UNA
           ECUACION"
210 PRINT "METODO DE TRASPOSICION"
215 PRINT
220 PRINT "LISTE LAS LINEAS 500 A 535"
225 PRINT "ESCRIBIR LA ECUACION EN LA LINEA 525"
230 PRINT "EN LA FORMA TRASPUESTA"
235 PRINT
240 PRINT "ESTIMACION INICIAL DE LA SOLUCION?";
245 INPUT XO
250 PRINT
255 PRINT "GRADO DE EXACTITUD?";
```

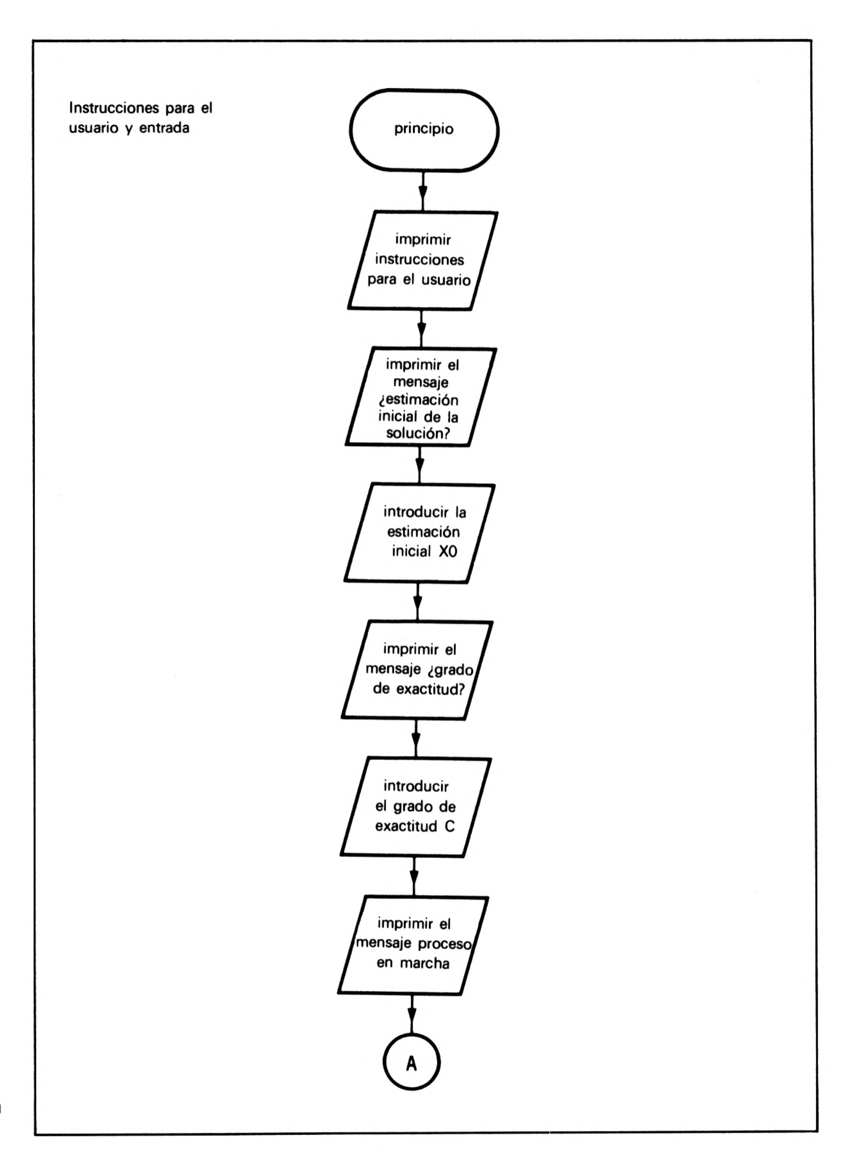
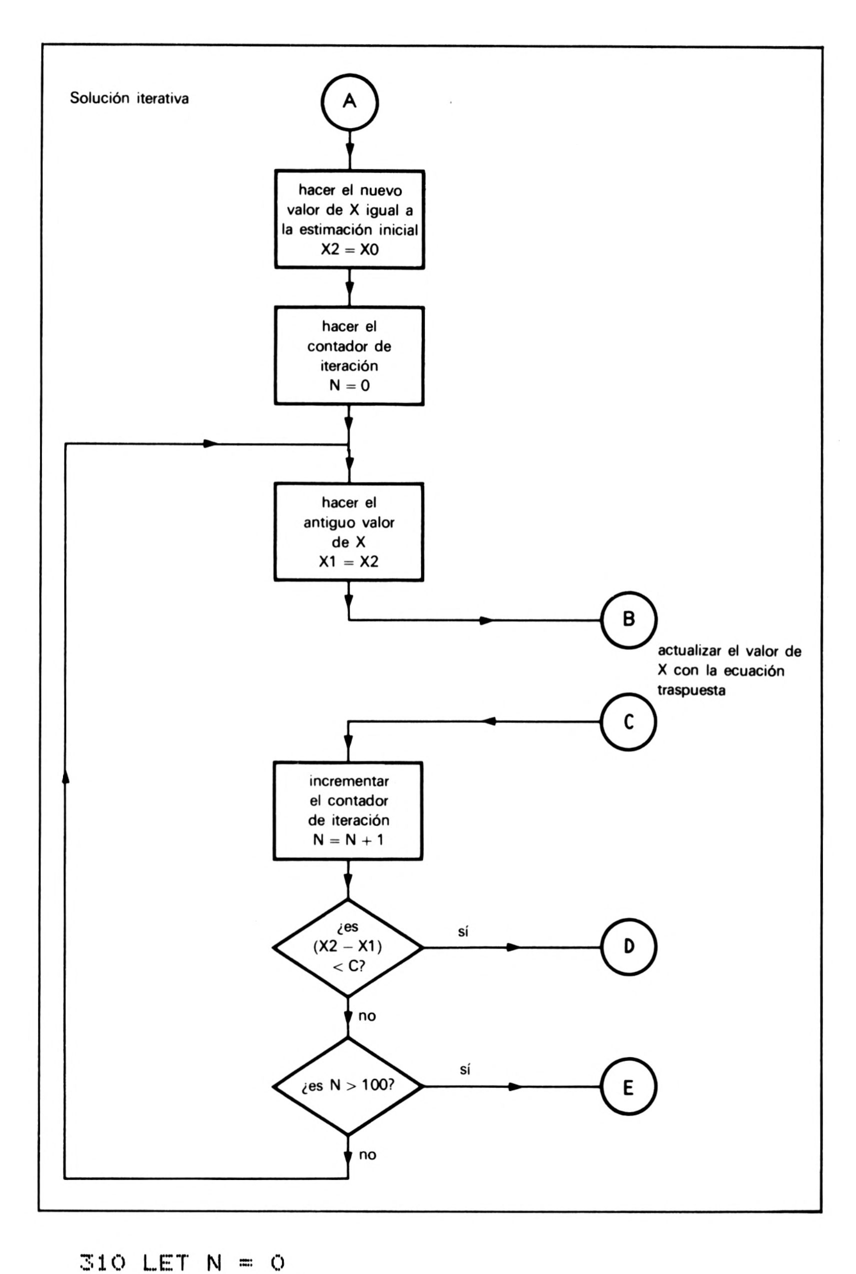


Figura 26.1
Diagrama de flujo del programa ejemplo 26.1

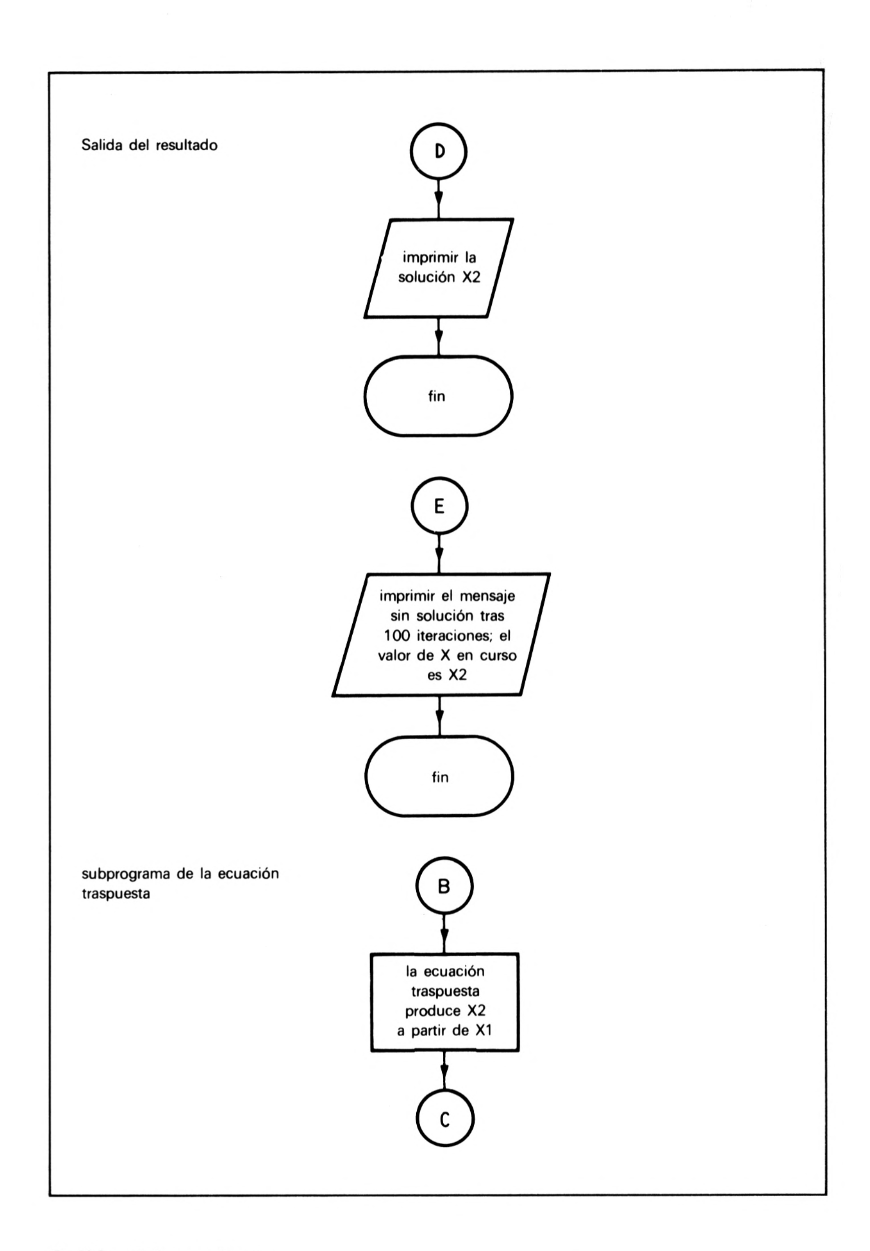
```
260 INPUT C
265 PRINT
270 PRINT "PROCESO EN MARCHA"
275 PRINT
280 REM
```

Solución iterativa

```
300 REM SOLUCION ITERATIVA
305 LET X2 = X0
```




```
310 LET N = 0
315 LET X1 = X2
320 GOSUB 500: REM OBTENCION DE X2 DE LA ECUA
                   CION TRASPUESTA
325 LET N = N + 1
330 IF ABS(X2 - X1) < C THEN 405
335 IF N > 100 THEN 420
340 GOTO 315
345 REM
```



Salida del resultado

```
400 REM SECCION DE SALIDA
405 PRINT "LA SOLUCION ES"; X2
410 PRINT
415 STOP
420 PRINT "SIN SOLUCION TRAS 100 ITERACIONES"
425 PRINT "EL VALOR DE X EN CURSO ES"; X2
430 PRINT
435 STOP
```

Subprograma de la ecuación traspuesta

```
 500 REM SUBPROGRAMA DE LA ECUACION TRASPUESTA
 505 REM PARAMETROS
 510 REM X1: VALOR ANTIGUO DE X
 515 REM X2: VALOR NUEVO DE X
 520 REM LA SIGUIENTE LINEA DEBE SER ESCRITA
 522 REM EN LA FORMA TRASPUESTA
 525 LET X2 = (X1 + 1) ^ 0.25
 530 RETURN
 535 REM
1000 END
```

Puntos de interés

- La ubicación del cálculo principal en un subprograma reduce la velocidad del proceso, pero facilita la modificación del programa.
- El mensaje de la línea 270 informa al usuario de que el proceso de solución está en marcha. Es un recurso útil, porque a veces el cálculo se prolonga durante mucho tiempo, y el mensaje reduce la probabilidad de que el usuario interrumpa el programa antes de que termine.
- Observe que las dos condiciones "repetir hasta que" se han situado juntas al final del bucle.
- El subprograma que contiene la ecuación traspuesta puede reemplazarse por una función definida.

26.4 Método de Gauss de resolución de sistemas de ecuaciones

En muchísimos cálculos científicos y de ingeniería, las soluciones a un problema se presentan en forma de **sistema de ecuaciones** con varias incógnitas. Un ejemplo de sistema sería:

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 2 \\ 5x_1 + 12x_2 + x_3 &= 32 \\ 3x_1 + 9x_2 + x_3 &= 24 \end{aligned}$$

En este caso se trata de un sistema de tres ecuaciones con tres incógnitas. Los números por los que se multiplican las incógnitas se llaman **coeficientes,** y **constantes** los que ocupan los miembros derechos de las ecuaciones. En general, puede haber sistemas con cualquier número de ecuaciones y el mismo número de incógnitas.

Algunos de estos sistemas no tienen solución, y otros tienen muchas. Aquí nos limitaremos a los sistemas "bien educados", con una solución única que es el conjunto de valores de las incógnitas que satisface todas las ecuaciones. Por razones de simplicidad, en los ejemplos haremos referencia únicamente a sistemas de tres ecuaciones con tres incógnitas.

Hay varias técnicas de resolución de sistemas de ecuaciones. La de **eliminación de Gauss** es muy sencilla, pero a la vez muy potente. No se trata de un número de aproximación, y funciona con sistemas de cualquier número de ecuaciones y el mismo número de incógnitas. Admite pruebas para interrumpir el proceso si el sistema carece de solución. En el método de eliminación de Gauss, los coeficientes y las constantes se consideran como elementos de una matriz; el proceso tiene dos fases:

> En primer lugar se restan sistemáticamente múltiplos de unas ecuaciones (o filas de matrices) a otras hasta crear una **matriz triangular** en la que todos los elementos situados por debajo de la diagonal principal sean nulos.
>
> En segundo lugar se procede a la **restitución,** partiendo de la fila inferior y avanzando hacia arriba, hasta obtener los valores de todas las incógnitas.

Apliquemos este método al ejemplo anterior:

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 2 \quad ① \\ 5x_1 + 12x_2 + x_3 &= 32 \quad ② \\ 3x_1 + 9x_2 + x_3 &= 24 \quad ③ \end{aligned}$$

El primer paso es eliminar los coeficientes segundo y tercero de la primera columna. Para ello se substrae a la segunda ecuación la primera multiplicada por 5, y a la tercera también la primera multiplicada por 3:

$$\begin{aligned} & & x_1 + 2x_2 - x_3 &= 2 \quad ① \\ ② - 5 \times ① & & 2x_2 + 6x_3 &= 22 \quad ④ \\ ③ - 3 \times ① & & 3x_2 + 4x_3 &= 18 \quad ⑤ \end{aligned}$$

A continuación se elimina el último coeficiente de la segunda columna restando a la tercera ecuación la segunda multiplicada por 3/2:

$$\begin{aligned} & & x_1 + 2x_2 - x_3 &= 2 \quad ① \\ & & 2x_2 + 6x_3 &= 22 \quad ④ \\ ⑤ - \frac{3}{2} \times ④ & & - 5x_3 &= -15 \quad ⑥ \end{aligned}$$

Ahora los coeficientes de las ecuaciones forman ya una matriz triangular. El proceso de restitución empieza, pues, por la tercera ecuación:

$$x_3 = \frac{-15}{-5} = 3$$

$$x_2 = \frac{22 - 6x_3}{2} = \frac{22 - 6 \times 3}{2} = 2$$

$$x_1 = \frac{2 - 2x_2 + x_3}{1} = \frac{2 - 2 \times 2 + 3}{1} = 1$$

La solución es, pues:

$$x_1 = 1, \ x_2 = 2, \ x_3 = 3.$$

26.5

Programa ejemplo 26.2

El objetivo de este programa es ejecutar el método de eliminación de Gauss sobre un sistema de tres ecuaciones con tres incógnitas como el estudiado en la sección anterior. Para poder escribir el programa, es preciso definir el problema en términos más generales. Los coeficientes y las constantes pasan a convertirse en elementos de una matriz, identificados mediante dos subíndices:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= a_{14} \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= a_{24} \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= a_{34} \end{aligned}$$

Como todas las operaciones se ejecutarán sobre los elementos de la matriz, pueden omitirse las incógnitas:

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{matrix}$$

Sea a_{ij} el elemento general de la matriz, situado en la fila i y la columna j. En general, la matriz tendrá n filas y $n + 1$ columnas.

El algoritmo de creación de la matriz triangular sobre la diagonal principal es el siguiente:

Para $k = 1$ hasta $n - 1$, repetir el proceso:

Para $i = k + 1$ hasta n, repetir el proceso:

Hacer el multiplicador $m = \frac{aik}{akk}$.

Para $j = k$ hasta $n + 1$, repetir el proceso:

Hacer $a_{ij} = a_{ij} - m \cdot a_{kj}$.

El proceso de restitución queda descrito por este otro algoritmo:

Para $k = n$ hasta 1 en pasos de -1, repetir el proceso:

Hacer el substraendo $s = 0$.

Para $j = k + 1$ hasta n, repetir el proceso:

Hacer $s = s + a_{kj} \cdot x_j$.

Hacer $x_k = \frac{a_{kn+1} - s}{a_{kk}}$.

Observe que la repetición del bucle interno ocurre cero veces si $k = n$.

Las estructuras de datos necesarias son dos matrices bidimensionales para los coeficientes y las constantes y otra unidimensional para las incógnitas. Las fases ya están especificadas con detalle suficiente como para pasar a escribir el programa. Sólo se ofrecerá aquí la sección de cálculo; los segmentos de entrada y salida quedan como ejercicios a cargo del lector.

Variables

A(3, 4)	Matriz de coeficientes y constantes.
X(3)	Matriz de incógnitas.
N	Número de incógnitas (3 en este caso).
I, J, K	Contadores de bucle.
M	Multiplicador.
S	Substraendo.

Diagrama de flujo

La figura 26.2 ilustra el flujo de control de los módulos de cálculo del programa ejemplo 26.2.

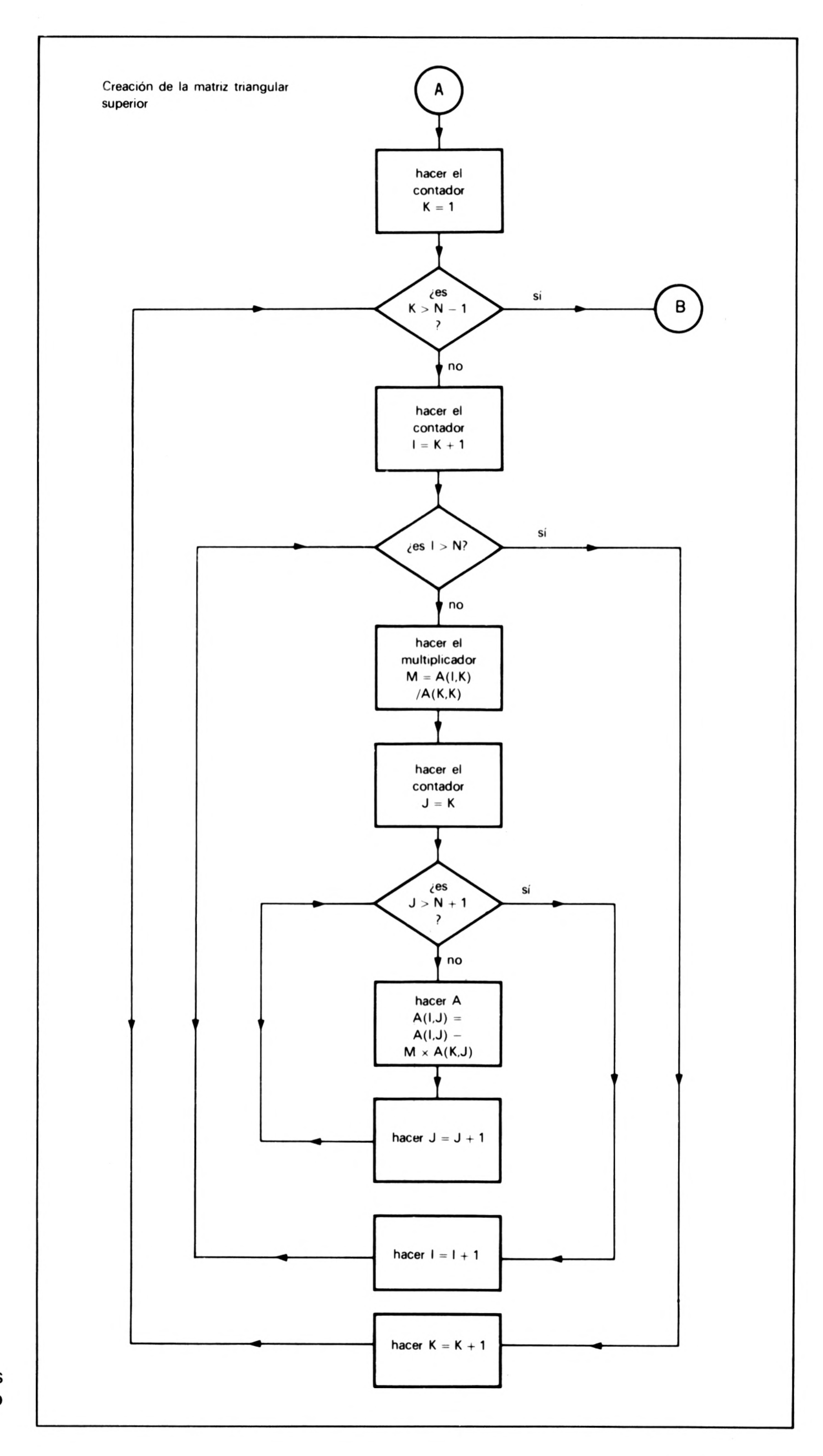
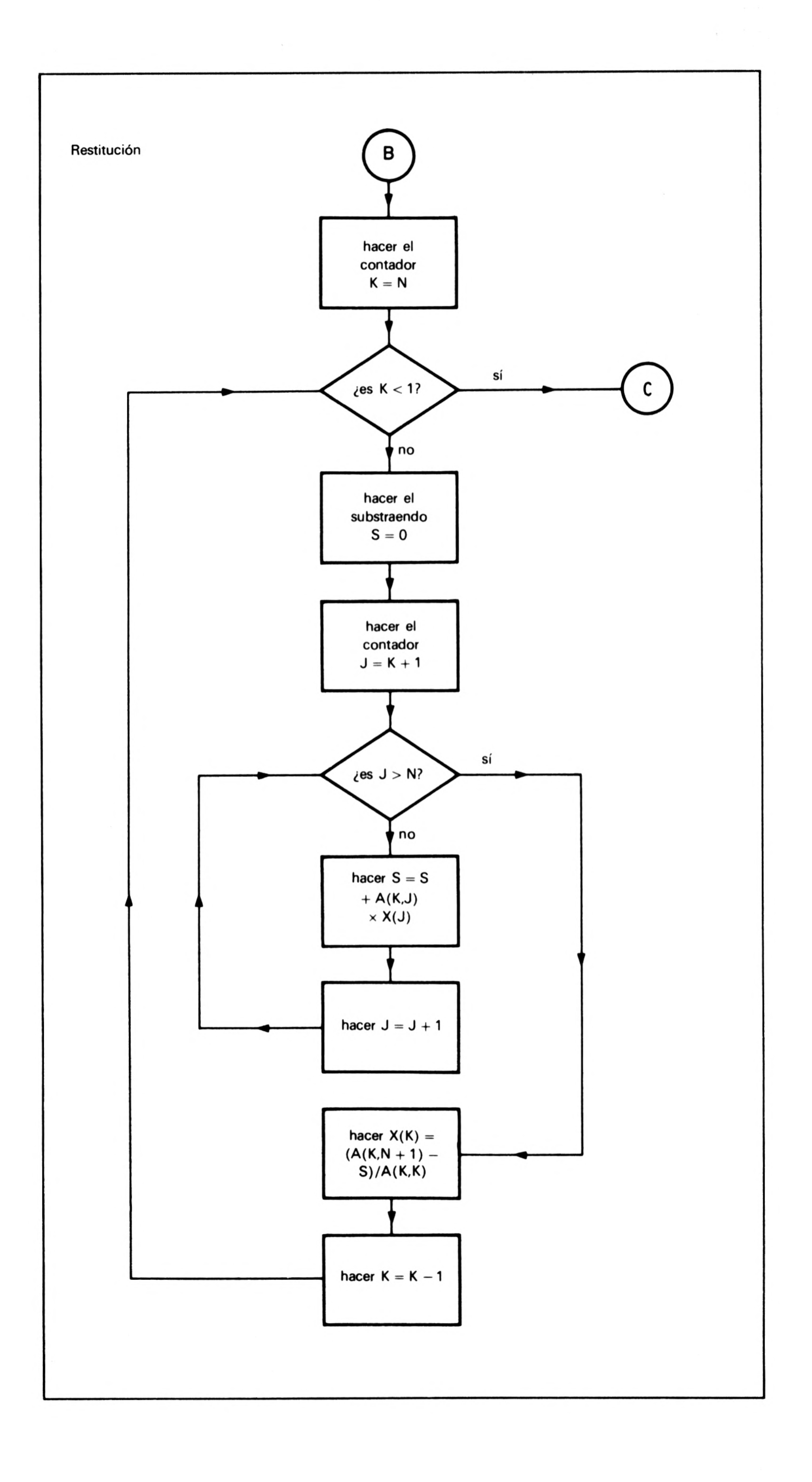


Figura 26.2
Diagrama de flujo de los módulos de proceso del programa ejemplo 26.2


Restitución
B
hacer el
contador
K = N
¿es K < 1?
sí
C
no
hacer el
substraendo
S = 0
hacer el
contador
J = K + 1
¿es J > N?
sí
no
hacer S = S
+ A(K,J)
× X(J)
hacer J = J + 1
hacer X(K) =
(A(K,N + 1) –
S)/A(K,K)
hacer K = K – 1

Programa

```
100 REM PROGRAMA EJEMPLO 26.2
105 REM SOLUCION DE UN SISTEMA DE ECUACIONES
110 REM POR EL METODO DE ELIMINACION DE GAUSS
115 DIM A(3, 4), X(3)
120 LET N = 3
125 REM
```

Módulo de entrada

```
200 REM EL MODULO DE ENTRADA
205 REM SE DEBE ESCRIBIR COMO UN EJERCICIO
210 REM
```

Creación de una matriz triangular sobre la diagonal principal

```
300 REM CREACION DE LA MATRIZ TRIANGULAR SUPE
        RIOR
305 FOR K =1 TO N - 1
310     FOR I = K + 1 TO N
315         LET M = A(I, K)/A(K, K)
320         FOR J = K TO N + 1
325             LET A(I,J) = A(I,J) - M * A(K,J)
330         NEXT J
335     NEXT I
340 NEXT K
345 REM
```

Restitución

```
400 REM RESTITUCION
405 FOR K = N TO 1 STEP -1
410     LET S = 0
412     IF K + 1 > N THEN 430
415     FOR J = K + 1 TO N
420         LET S = S + A(K, J) * X(J)
425     NEXT J
430     LET X(K) = (A(K, N + 1) - S)/A(K, K)
435 NEXT K
440 REM
```

Módulo de salida

```
 500 REM EL MODULO DE SALIDA
 505 REM SE DEBE ESCRIBIR COMO UN EJERCICIO
 510 REM
1000 END
```

Puntos de interés

- Las líneas de programa que se repiten están **indentadas**, para que se vea claramente la relación entre los bucles internos y resulte fácil localizar el principio y el final de cada uno. Por desgracia, en muchos ordenadores el sangrado se pierde al visualizar o imprimir el programa, porque se ignoran los espacios redundantes; por eso en este libro no se ha empleado mucho este recurso.
- El número de ecuaciones que han de resolverse se determina en la línea 120. Si ésta y la instrucción **DIM** se modifican, el programa podrá aceptar sistemas de más ecuaciones con más incógnitas.
- Algunos ordenadores ejecutan los bucles **FOR ... NEXT** al menos una vez. En tal caso, el contenido entre las líneas 415 y 425 debería ir precedido por una instrucción que lo salte si **K** es igual a **N**.

26.6

Integración numérica

El proceso de **integración** de una función puede considerarse como el proceso de cálculo del área situada bajo la curva de esa función, tal como ilustra la figura 26.3. Observe que la integración tiene lugar entre un intervalo definido de valores que constituyen los **límites de la integración.**

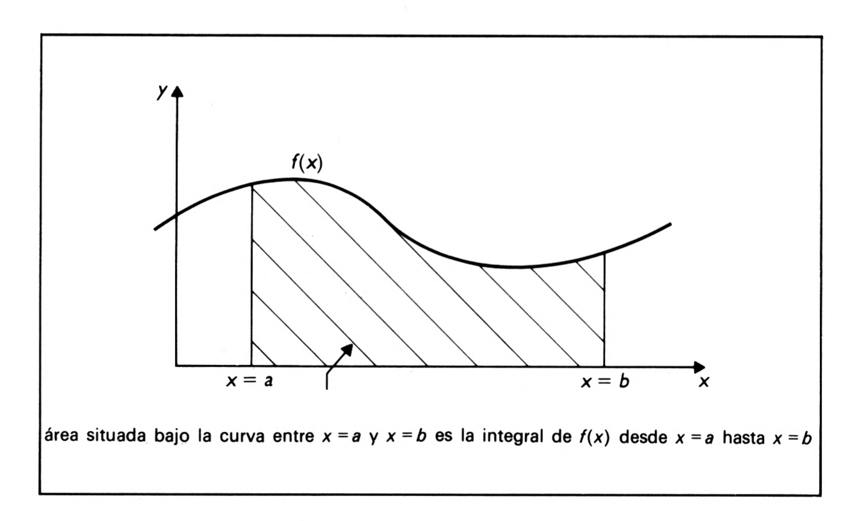


Figura 26.3
Integración

Para muchísimas funciones, la integración es un proceso difícil, y son numerosas las que no pueden integrarse exactamente de ninguna forma. Se llama integración numérica al proceso de obtención del valor aproximado de una integral por medio de una secuencia de cálculos relativamente sencillos. Todo lo necesario para ello es la fórmula de la función que ha de integrarse o, en su defecto, los valores que va tomando esa función a intervalos regulares a lo largo del espacio de integración.

La **regla de los trapecios** constituye una técnica de integración sencilla y potente. El método consiste en ir aproximándose a la curva de la función por medio de una serie de rectas tendidas entre puntos igualmente espaciados de dicha curva; de esta forma, el área situada bajo la curva se descompone en una serie de **trapecios** (de ahí el nombre del método).

La figura 26.4 ilustra el proceso de división del área definida por la curva en n trapecios de longitud h. Los valores de la función son f_0, $f_1, ..., f_n$. El valor de h viene dado por la ecuación

$$h = \frac{h - a}{n}.$$

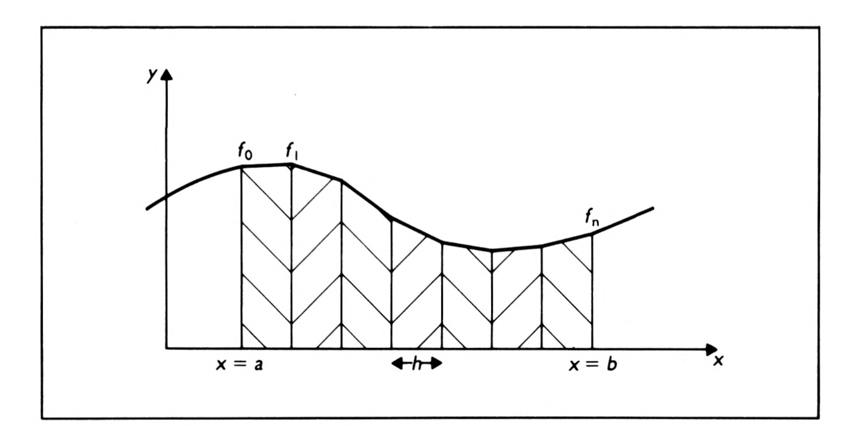


Figura 26.4
Regla de los trapecios

Por tanto, la fórmula para determinar el área definida por la curva subdividida de la forma descrita, será:

$$I = \frac{h}{2}[f_0 + f_n + 2(f_1 + f_2 + \cdots + f_{n-1})]$$

26.7

Programa ejemplo 26.3

El objetivo de este programa es ejecutar la regla de integración del trapecio descrita en la sección anterior. Son entradas del programa la función que ha de integrarse, los límites de la integración y el número de subdivisiones del área.

Diseño del programa

El programa tiene una estructura entrada-proceso-salida muy sencilla. La función que ha de integrarse está contenida en un subprograma, porque así es más fácil modificar el programa. Dado que la única parte de éste con cierta complejidad es el bucle encargado de acumular la suma de la fórmula de integración, parece razonable pasar directamente a escribir las sentencias que lo constituyen.

Variables

A, B	Límites de integración.
F	Valor de la función que ha de integrarse, parámetro del subprograma.
X	Variable independiente, parámetro del subprograma.
N	Número de subdivisiones del área definida por la curva.
H	Anchura del trapecio.
I	Integral.
F1, F2	Valores de la función en los límites de la integral.
S	Suma acumulada de la fórmula de integración.
K	Contador de bucle.

Diagrama de flujo

Véase la figura 26.5.

Programa

```
100 REM PROGRAMA EJEMPLO 26.3
105 REM INTEGRACION POR MEDIO DE LA REGLA DE LOS
        TRAPECIOS
110 REM
```

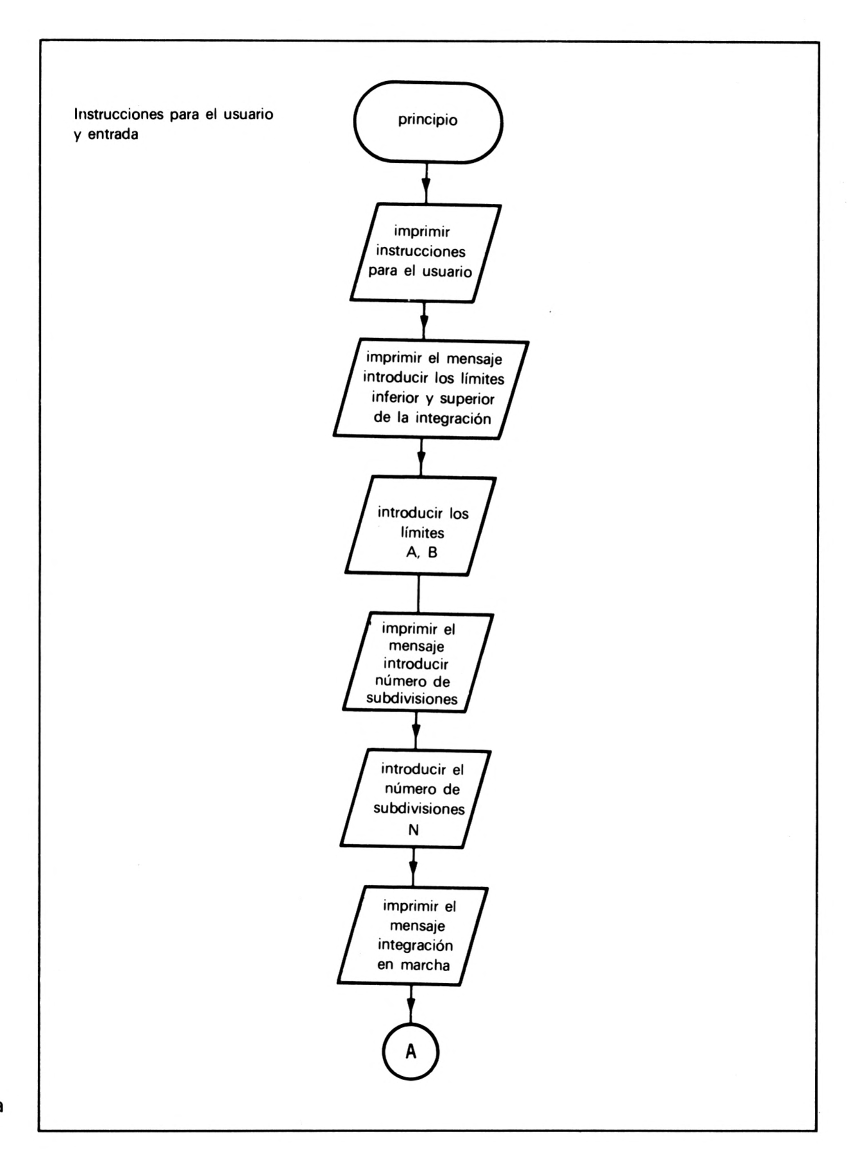
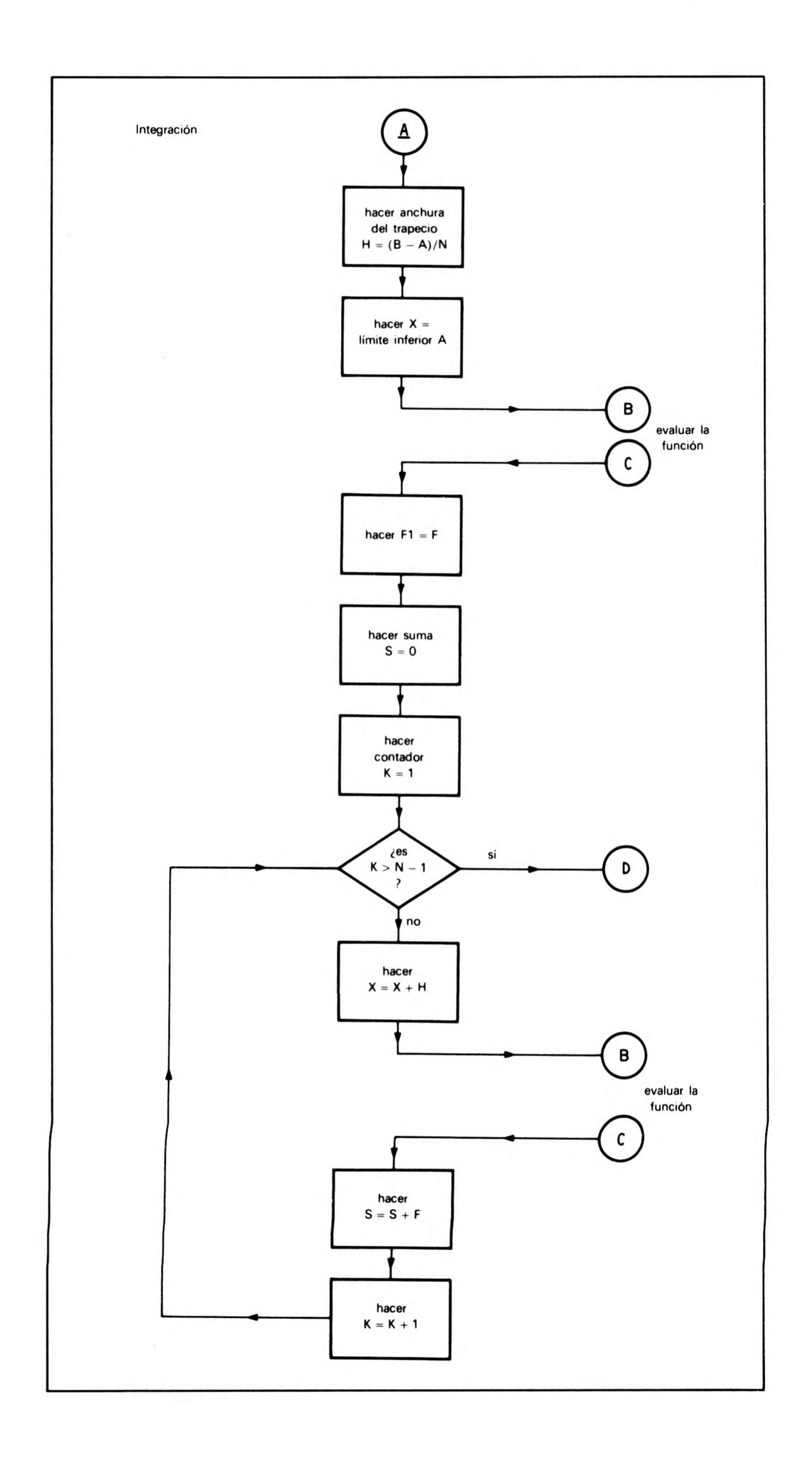
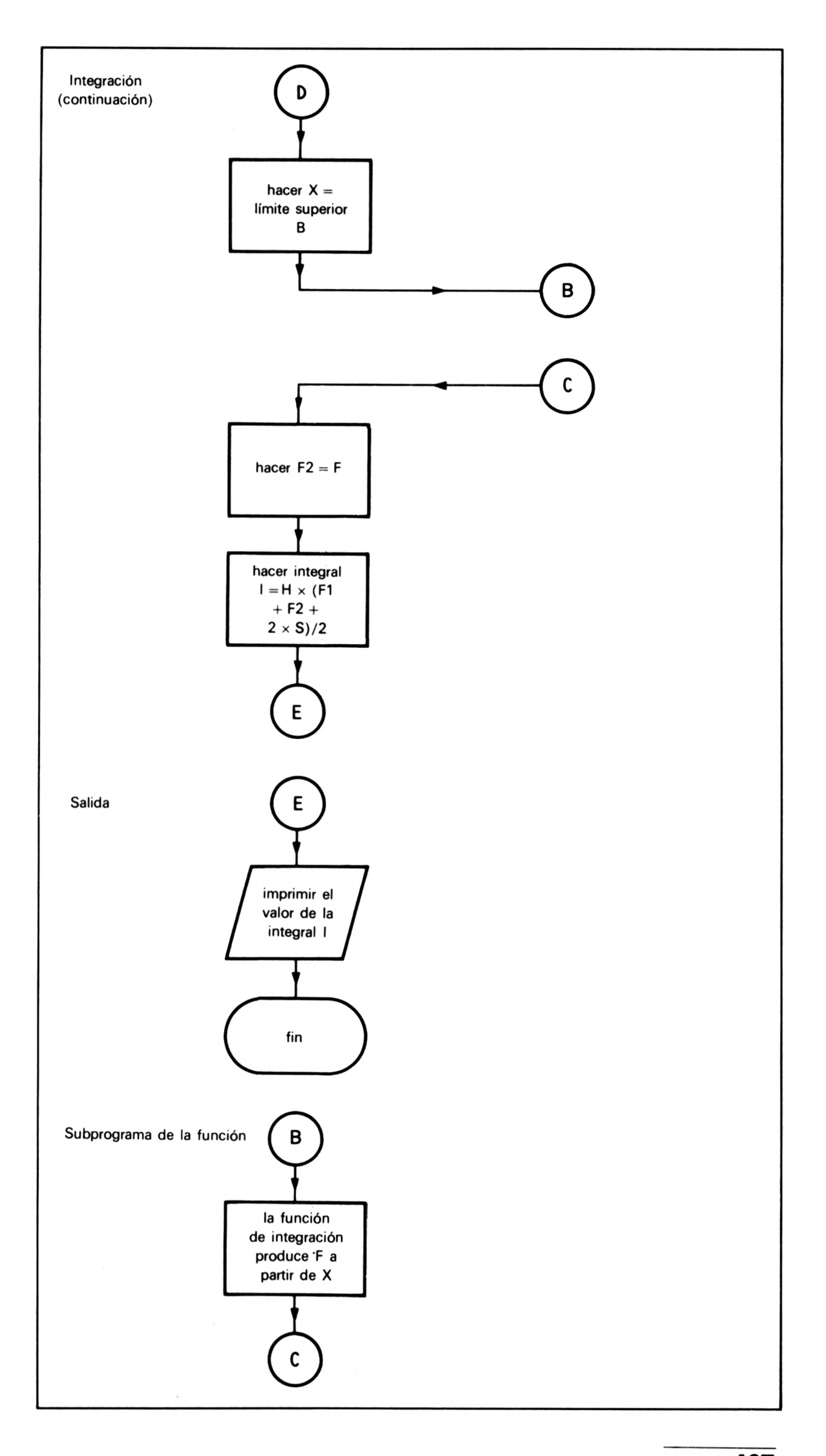


Figura 26.5
Diagrama de flujo del programa ejemplo 26.3

Instrucciones para el usuario y entrada

```
200 REM INSTRUCCIONES DEL USUARIO E INTRODUCCION
        DE DATOS
205 PRINT "INTEGRACION POR MEDIO DE LA REGLA DE
           LOS TRAPECIOS"
210 PRINT
215 PRINT "VISUALIZACION DE LAS LINEAS 500 A 535
           "
220 PRINT "ESCRIBIR LA FUNCION EN LA LINEA 525"
```


Integración
A
hacer anchura
del trapecio
H = (B − A)/N
hacer X =
límite inferior A
B
evaluar la
función
C
hacer F1 = F
hacer suma
S = 0
hacer
contador
K = 1
¿es
K > N − 1
?
sí
D
no
hacer
X = X + H
B
evaluar la
función
C
hacer
S = S + F
hacer
K = K + 1


Integración
(continuación)
D
hacer X =
límite superior
B
B
C
hacer F2 = F
hacer integral
I = H × (F1
+ F2 +
2 × S)/2
E
Salida
E
imprimir el
valor de la
integral I
fin
Subprograma de la función
B
la función
de integración
produce F a
partir de X
C

```
225 PRINT
230 PRINT "INTRODUCIR LOS LIMITES INFERIOR Y SU
         PERIOR"
235 PRINT "DE INTEGRACION"
240 INPUT A, B
245 PRINT
250 PRINT "INTRODUCIR EL NUMERO DE SUBDIVISIONES
         "
255 PRINT "DEL AREA DE INTEGRACION"
260 INPUT N
265 PRINT
270 PRINT "INTEGRACION EN MARCHA"
275 PRINT
280 REM
```

Integración

```
300 REM INTEGRACION
305 LET H = (B - A)/N
310 LET X = A
315 GOSUB 500
320 LET F1 = F
325 LET S = 0
330 FOR K = 1 TO N - 1
335 LET X = X + H
340 GOSUB 500
345 LET S = S + F
350 NEXT K
355 LET X = B
360 GOSUB 500
365 LET F2 = F
370 LET I = H * (F1 + F2 + 2 * S)/2
375 REM
```

Salida

```
400 REM SALIDA
405 PRINT "VALOR DE LA INTEGRAL:"; I
410 STOP
415 REM
```

Subprograma de la función

```
500 REM SUBPROGRAMA DE LA FUNCION A INTEGRAR
505 REM PARAMETROS
510 REM X: VARIABLE INDEPENDIENTE
515 REM F: VALOR DE LA FUNCION
520 REM LA FUNCION SE DEBE INTRODUCIR EN LA
        SIGUIENTE LINEA
```

```
525 LET F = (X * X + 1)/(X * X - 1)
530 RETURN
535 REM
600 END
```

Puntos de interés

- El subprograma que contiene la función cuya integral quiere calcularse puede también escribirse en forma de función definida por el usuario.
- El programa sigue la disposición general del estudiado en el ejemplo 26.1.

26.8 Empleo del análisis numérico

Aunque a primera vista puede no parecer así, la dificultad principal que ofrece el sistema numérico no está en resolver las ecuaciones, sino en plantearlas.

Antes de poder aplicar un proceso matemático a una situación determinada, es preciso estudiar ésta con detenimiento y crear un **modelo matemático** que la reproduzca. Se llama así a la determinación de los aspectos de la situación que han de medirse más las ecuaciones encargadas de relacionar esas medidas entre sí. A continuación se resuelven aquéllas, por lo general mediante una técnica de análisis numérico.

Crear un modelo matemático es casi siempre mucho más difícil que resolver las ecuaciones a que da lugar. Si el modelo no es adecuado, los resultados producidos por él serán inútiles. En el próximo capítulo, titulado "Simulación", estudiaremos la elaboración de modelos.

26.9 Conclusión

En este capítulo hemos expuesto brevemente algunas técnicas de análisis numérico y hemos desarrollado un par de programas para llevarlas a la práctica. También se han dado algunas ideas someras sobre el empleo del análisis numérico. Las conclusiones finales más importantes se resumen en:

- Se llama análisis numérico a una serie de técnicas que permiten resolver diversos problemas matemáticos mediante secuencias repetitivas de cálculos.
- Las técnicas iterativas se materializan en la realización de una serie de cálculos sencillos que convergen hacia un resultado; éste puede obtenerse con cualquier grado de exactitud deseado.
- El método de trasposición es una técnica iterativa de aproximación que permite resolver ecuaciones muy diversas.
- El método de eliminación de Gauss es un procedimiento de tipo general de resolución de sistemas de varias ecuaciones con varias incógnitas.
- La regla de los trapecios constituye un método de integración aproximativo muy sencillo.

26 Ejercicio

1. Defina brevemente los siguientes términos: análisis numérico, técnica iterativa, eliminación de Gauss, regla de los trapecios, convergencia, matriz triangular sobre la diagonal principal, modelo matemático.
2. Escriba los módulos de entrada y salida del programa ejemplo 26.2.
3. Vuelva a escribir los subprogramas de los ejemplos 26.1 y 26.3 en forma de funciones definidas por el usuario. Las definiciones deben situarse cerca del comienzo de los programas, antes de que se utilicen las funciones. Modifique las llamadas a los subprogramas para adaptarlas a las funciones.
4. Utilice el programa ejemplo 26.1 para resolver las siguientes ecuaciones:

   a) $x^3 - 2x = 0$ estimación próxima: $x = 1$

   b) $x^2 + \frac{1}{x^2} - 3 = 0$ estimación próxima: $x = 1$

   c) $5 \log x - x = 0$ estimación próxima: $x = 10$ (log natural)

   d) $e^x - x - 1 = 0$ estimación próxima: $x = 1{,}5$

   e) $\frac{x}{x+1} + \frac{x}{x-1} = 2$ estimación próxima: $x = 3$

5. Un método iterativo de resolución de ecuaciones que en la mayor parte de los casos produce raíces que convergen más rápidamente que las del método de trasposición es el llamado **método de Newton.** Sea la ecuación

   $$f(x) = 0$$

   La derivada del miembro izquierdo puede representarse por $f'(x)$. Si x_n es una estimación de una solución de la ecuación, el método de Newton produce una estimación más exacta x_{n+1} mediante la fórmula:

   $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Esta fórmula puede utilizarse en lugar de la ecuación traspuesta del programa ejemplo 26.1.

a) Modifique el programa ejemplo 26.1 de forma que indique a la salida el número de iteraciones necesarias para producir una solución con el grado de exactitud deseado.

b) Resuelva algunas o todas las ecuaciones de la pregunta 4 con una precisión de 0,001 utilizando el programa ejemplo 26.1 y anotando en cada caso el número de iteraciones realizadas.

c) Calcule las diferenciales de los miembros izquierdos de las ecuaciones de la pregunta 4 y determine la fórmula de Newton que corresponde a cada una de ellas.

d) Incorpore el método de Newton al programa ejemplo 26.1 y resuelva las ecuaciones a que estamos haciendo referencia, anotando siempre el número de iteraciones realizadas.

e) Comente los resultados.

6. Utilice el programa ejemplo 26.2 para resolver los siguientes sistemas de ecuaciones:

a)
$$\begin{aligned} 6x_1 + 3x_2 + 4x_3 &= 23 \\ x_1 - 2x_2 - 3x_3 &= -5 \\ 9x_1 - x_2 + 2x_3 &= 11 \end{aligned}$$

b)
$$\begin{aligned} 10x_1 + 3x_2 + x_3 &= 18 \\ 2x_1 + 8x_2 + 2x_3 + x_4 &= 15 \\ x_2 + 14x_3 + 3x_4 + x_5 &= 24 \\ 4x_3 + 12x_4 + 6x_5 &= 20 \\ x_4 + 9x_5 &= 10 \end{aligned}$$

c)
$$\begin{aligned} 3{,}542x_1 + 9{,}375x_2 - 2{,}633x_3 &= 18{,}394 \\ 1{,}628x_1 + 12{,}348x_2 + 0{,}493x_3 &= 12{,}502 \\ 0{,}631x_1 - 2{,}517x_2 + 8{,}519x_3 &= 4{,}718 \end{aligned}$$

7. El método de eliminación de Gauss para resolver sistemas de ecuaciones puede ampliarse para generar matrices inversas. En lugar de partir de la columna de constantes de los miembros derechos de las ecuaciones, se escribe una matriz unidad de dimensiones adecuadas. Por ejemplo:

$$\begin{matrix} 3 & 4 & 2 & 1 & 0 & 0 \\ 1 & 6 & 3 & 0 & 1 & 0 \\ 2 & 1 & 9 & 0 & 0 & 1 \end{matrix}$$

La matriz triangular superior se produce de la forma habitual y los cálculos se amplían a la prolongación de las filas. La restitución se repite una vez para cada columna de cifras del lado derecho de la matriz: cada repetición genera una columna de la matriz inversa.

Corrija el programa ejemplo 26.2 para que pueda calcular matrices inversas. Si lo desea, puede escribir la versión corregida como un módulo del programa ejemplo 7.2.

8. Utilice el programa escrito como respuesta a la pregunta 7 para calcular las inversas de las siguientes matrices:

a)
$$\begin{matrix} 3 & 5 & 9 \\ 2 & 4 & 1 \\ 3 & 0 & 6 \end{matrix}$$

b)
$$\begin{matrix} 0{,}707 & -0{,}707 & 0 \\ 0{,}707 & 0{,}707 & 0 \\ 0 & 0 & 1 \end{matrix}$$

c)

10	3	5	1	0
4	8	2	2	0
0	3	9	5	2
0	0	2	8	1
0	0	1	5	12

9. Utilice el programa ejemplo 26.3 para integrar las siguientes funciones:

a) x^2 entre $x = 0$ y $x = 1$
b) $x^3 + 2x^2 + 5x - 3$ entre $x = 1$ y $x = 2$
c) $\dfrac{x + 1}{x - 1}$ entre $x = 2$ y $x = 3$
d) $\log x$ entre $x = 0$ y $x = 2{,}71828$
e) $1/x$ entre $x = 1$ y $x = 2$

Utilice primero un valor para h igual a un décimo del intervalo de integración y a continuación repita el cálculo con un nuevo valor de h igual a un décimo del anterior. Comente los resultados.

10. El método numérico de integración conocido como **regla de Simpson** se aproxima a la función que ha de integrarse por medio de una serie de curvas parabólicas. El resultado es la siguiente fórmula de integración (las variables son las del programa ejemplo 26.3):

$$I = \frac{h}{3}[f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 4f_{n-1} + f_n]$$

La única condición para aplicar esta regla es que n sea par.

a) Modifique el programa ejemplo 26.3 para que realice la regla de Simpson.
b) Integre las funciones de la pregunta 9 por este procedimiento, utilizando los mismos valores de h que entonces.
c) Compare los resultados obtenidos por los dos procedimientos y coméntelos.


RESET
RETURN

27 Simulación

La **simulación** constituye una de las áreas más útiles del cálculo científico. En este capítulo veremos algunas ideas generales relacionadas con ella, y en concreto expondremos el concepto de **modelo matemático,** acompañándolo de un ejemplo resuelto bastante amplio que se desarrollará aún más en el ejercicio.

Utilizaremos aquí algunas de las técnicas matemáticas descritas en el capítulo anterior, entre ellas la **iteración,** y definiremos además el concepto de **ecuación diferencial.** Como viene siendo habitual, se hará uso de las técnicas de diseño y estructuración de programas ya conocidas.

Este capítulo es, pues, de naturaleza más decididamente matemática que la mayoría de los demás, y puede omitirse sin pérdida de continuidad.

27.1 Simulación: elaboración de un modelo

En el sentido más general de la palabra, simular es hacer que un sistema se comporte como otro. Aplicada a los ordenadores, la idea de simulación se hace más concreta, y hace referencia a un ordenador

que se conduce de forma que imita el comportamiento de ciertos aspectos de otro sistema.

Repare en la expresión "ciertos aspectos". La mayor parte de los sistemas reales son demasiado complejos y se conocen demasiado poco como para poder simularlos con detalle. Todo lo que puede hacerse es, por tanto, simular el comportamiento de los aspectos esenciales o necesarios de esos sistemas, y de aquí surge la idea de **modelo.**

Un modelo es una simplificación deliberada y organizada de un sistema que se utiliza para identificar las pautas de comportamiento esenciales de éste. Si consta de ecuaciones, se habla de **modelo matemático.** Tales ecuaciones se usan para investigar el comportamiento del sistema, tanto bajo condiciones conocidas o presentes como bajo desconocidas o futuras. Además, si el sistema está controlado de alguna forma, el modelo permite investigar los efectos de los diversos niveles de control.

La simulación se usa en situaciones muy variadas, como procesos físicos, químicos y biológicos, sistemas económicos, estrategia militar y gran cantidad de aplicaciones de ingeniería. Las razones que conducen a la simulación se encuentran casi siempre entre las siguientes:

- Un proceso muy lento puede simularse con un modelo rápido, y viceversa.
- El sistema real es demasiado peligroso como para experimentar directamente sobre él.
- La simulación se lleva a cabo antes de la construcción del sistema real.
- La simulación da una idea del funcionamiento del sistema y puede emplearse como medio de adiestramiento.
- La simulación con ordenador es mucho más barata que cualquier otro procedimiento de investigación comparable.

27.2

Proceso de creación de un modelo

El camino que va desde el sistema real —inevitablemente complejo y poco conocido— hasta un modelo sencillo y exacto no es nada fácil. Es imprescindible avanzar con precaución y tener siempre en cuenta los objetivos del ejercicio de simulación.

Se han propuesto varios procedimientos para encarar el modelado de sistemas. El descrito a continuación es bastante representativo, y exige poco más que pensamiento sistemático y sentido común:

- Determinar los aspectos relevantes del sistema en estudio, prestando especial atención a los que pueden medirse.

- Formular ecuaciones que relacionen los aspectos relevantes mensurables y reflejen así el comportamiento del mismo.
- Resolver las ecuaciones, generalmente mediante alguna técnica numérica. Las soluciones iterativas son particularmente útiles, porque permiten simular directamente las variaciones temporales del sistema, a veces incluso en tiempo real.
- Compare el comportamiento del modelo definido mediante las ecuaciones con el sistema real. Lamentablemente, esta operación no siempre es posible, sobre todo si el primero se construye antes que el segundo.
- Utilice el modelo para investigar el comportamiento del sistema bajo condiciones diferentes, incluyendo posibles circunstancias futuras. Si el sistema dispone de algún medio de control, puede también investigarse su comportamiento a diferentes niveles y pautas de control.

En la práctica, estos pasos raramente se aplican en una secuencia ininterrumpida. Lo normal es que se aprecien en el modelo una serie de deficiencias y que simultáneamente aquél vaya **refinándose** mediante la repetición de uno o más pasos. Por otra parte, los puntos expuestos no deben considerarse como normas rigurosas, sino como pautas generales que se aplican con un criterio flexible en función de las circunstancias.

27.3 Características deseables en un modelo

El modelo debe reflejar con precisión los aspectos fundamentales del comportamiento del sistema y además conservar el "tacto" del mismo.

Debe también ser susceptible de conversión por refinamiento en otro modelo más perfecto si fuese necesario. Tiene que iluminar de una forma u otra el comportamiento o el funcionamiento del sistema. Y por último, sin dejar de satisfacer todas las condiciones mencionadas, el modelo debe ser lo más sencillo posible.

La esfera que se usa para representar la tierra, por ejemplo, es un modelo que tiene muchas de las características descritas: aunque no constituye una representación exacta, resulta útil a casi todos los efectos y es extraordinariamente simple.

Hay que insistir en que las virtudes mencionadas son difíciles de obtener en la práctica. Sólo un enfoque riguroso y sistemático del proceso de modelado llevará a la consecución de un modelo aceptable.

27.4

Un modelo de población

Dedicaremos esta sección a crear un modelo capaz de representar uno de los aspectos más característicos del comportamiento de cualquier especie animal: el incremento o la disminución del número de individuos que la componen. El modelo puede aplicarse tanto a la especie completa como a cualquier grupo definible dentro de la misma. En la próxima sección pondremos a prueba el modelo mediante el oportuno programa. Sólo hay dos cantidades numéricas: el número de individuos de la población y el tiempo.

La primera suposición que haremos dice que la tasa de crecimiento de la población es proporcional al número de individuos de la misma en un momento dado. La matemática de esta hipótesis es:

$$\frac{\mathrm{d}P}{\mathrm{d}t} = KP$$

donde P es el número de individuos de la población, t es el tiempo, $\mathrm{d}P/\mathrm{d}t$ es la tasa de crecimiento y K el **factor de crecimiento.**

Una ecuación como la anterior se llama **ecuación diferencial,** y puede resolverse de varias formas. Dado que va a utilizarse un ordenador, lo mejor es escribir una versión aproximativa de la ecuación, como

$$\frac{\Delta P}{\Delta t} = KP \qquad (\Delta \text{ es delta})$$

o

$$\Delta P = KP\,\Delta t$$

donde Δt es un intervalo finito de tiempo (en muchos casos, un año) y ΔP es el cambio experimentado por la población en ese intervalo.

El paso siguiente en el refinamiento del modelo afectará al factor de crecimiento K. Si K es una constante, ΔP se incrementará con P, y la población aumentará a un ritmo creciente. Se sabe que en ciertas circunstancias algunas poblaciones crecen muy rápidamente, pero en la práctica siempre alcanzan un tamaño límite.

Por tanto, parece razonable suponer que K disminuye a medida que la población crece. Este hecho vendría representado por la ecuación

$$K = A - BP$$

donde A y B son constantes.

La constante A es fácil de explicar. Si la población es muy pequeña, K es aproximadamente igual a A que, por tanto, puede considerarse como el factor de crecimiento en condiciones ideales.

La constante B está relacionada con presiones ambientales, como la competencia por el alimento, que limitan el crecimiento de la población. Si ésta tiene un tamaño límite L, el factor de crecimiento será cero cuando lo alcance. En otras palabras:

$$0 = A - BL$$

o bien

$$B = \frac{A}{L}$$

Dado que el significado de A y L es más evidente que el de B, conviene utilizar sus valores en lugar de los de ésta en las anteriores ecuaciones, operación que da lugar al modelo de población definitivo:

$$K = A - \frac{A}{L}P$$

o bien

$$K = A\left(1 - \frac{P}{L}\right)$$

Por tanto,

$$\Delta P = A\left(1 - \frac{P}{L}\right)P\Delta t.$$

Esta última ecuación sienta las bases del modelo. Relaciona la variación de tamaño de la población ΔP con la población actual P, un intervalo de tiempo fijo Δt y dos **parámetros:** la tasa de crecimiento en condiciones ideales A y el tamaño límite de la población L.

La ecuaciión del modelo se utiliza como sigue:

- Se miden o estiman los valores de A y L para una situación particular.
- Se elige un intervalo de tiempo adecuado (por lo general, $\Delta t = 1$ año o 10 años).
- Se mide o se elige un tamaño inicial de la población apropiado.
- Se sustituyen estos valores por las letras que los representan en la parte derecha de la ecuación del modelo y se calcula ΔP.

- Se suma este valor al actual de la población para obtener el tamaño de la misma al final del período considerado.
- La nueva cifra de población se utiliza para calcular el tamaño que alcanzará tras un nuevo intervalo.
- El proceso descrito puede extenderse sobre tantos períodos de tiempo como se desee.

27.5

Programa ejemplo 27.1

El objetivo de este programa es ejecutar en la práctica el modelo de población desarrollado en la sección anterior.

Diseño del programa

La estructura general del mismo es:

Introducir los valores de los parámetros, el valor inicial de la población, el intervalo de tiempo y la duración de la simulación.
Simular el comportamiento de la población a lo largo del período propuesto y presentar como resultado una tabla o una curva de su evolución.

El programa está escrito para obtener los resultados en una tabla (la opción de la curva queda como ejercicio para el lector). El primer paso del programa es bastante sencillo. El segundo puede detallarse algo más:

Mientras dure la simulación,
Repetir el proceso:

Presentar el tiempo transcurrido y el nivel de la población en ese momento.
Calcular el cambio de tamaño de la población.
Sumar ese cambio al tamaño de la población.

Observe que, dentro del bucle, el tiempo y el tamaño de la población se llevan a la salida antes de ser actualizados. El programa ya está explicado con minuciosidad suficiente y puede escribirse.

Variables

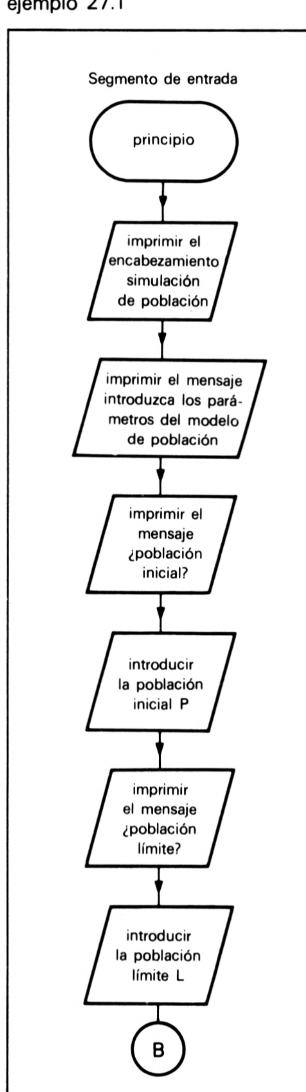
P Tamaño de la población y valor inicial a la entrada.
A Tasa de crecimiento en condiciones ideales.

L Tamaño límite de la población.
T1 Intervalo de tiempo de los pasos de la simulación ($=\Delta T$).
T2 Duración total de la simulación.
P1 Cambio experimentado por la población ($=\Delta P$).
T Tiempo, contador del bucle.

Diagrama de flujo

Véase la figura 27.1.

Figura 27.1
Diagrama de flujo del programa ejemplo 27.1



Programa

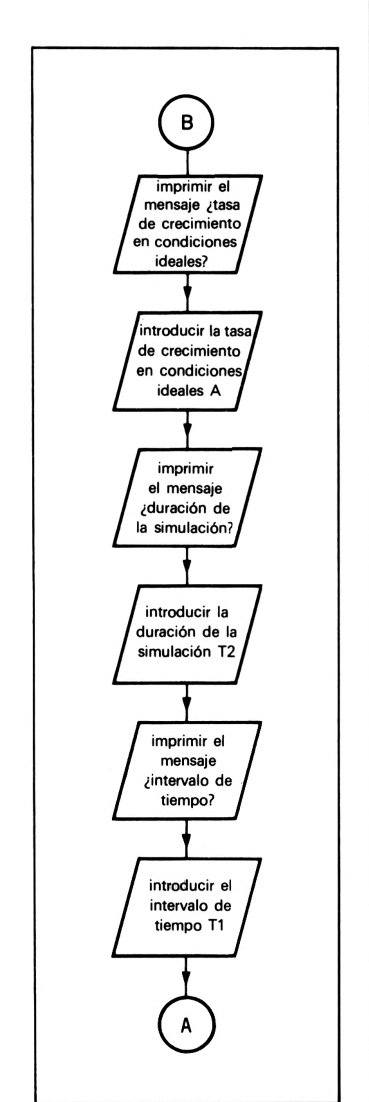
```
100 REM PROGRAMA EJEMPLO 27.1
105 REM SIMULACION DE POBLACION
110 REM
```

Segmento de entrada

```
200 REM SEGMENTO DE ENTRADA
205 PRINT "SIMULACION DE POBLACION"
210 PRINT
215 PRINT "INTRODUZCA LOS PARAMETROS DEL MODELO
          DE POBLACION"
220 PRINT
225 PRINT "POBLACION INICIAL?";
230 INPUT P
235 PRINT "POBLACION LIMITE?";
240 INPUT L
245 PRINT "TASA DE CRECIMIENTO EN CONDICIONES
          IDEALES?";
250 INPUT A
255 PRINT "DURACION DE LA SIMULACION?";
260 INPUT T2
265 PRINT "INTERVALO DE TIEMPO?"
270 INPUT T1
275 PRINT
280 PRINT
285 REM
```

Segmento de cálculo y tabulación

```
500 REM SEGMENTO DE CALCULO Y TABULACION
505 PRINT TAB(5); "TIEMPO"; TAB(15); "CRECIMIEN
       TO"; TAB(35); "POBLACION"
510 PRINT
515 LET P1 = 0
520 FOR T = 0 TO T2 STEP T1
525 PRINT TAB(5); T; TAB(15); P1; TAB(35); P
```



Segmento de cálculo y tabulación

A

imprimir los encabezamientos crecimiento de una población a lo largo del tiempo

hacer crecimiento P1 = 0

hacer contador T = 0

¿es T > T2?

sí

fin

no

imprimir el tiempo T, el crecimiento P1 y la población P

calcular el crecimiento P1 = A × (1 − P/L) × P × T1

actualizar la población P = P + P1

hacer T = T + T1

```
530 LET P1 = A * (1 - P/L) * P * T1
535 LET P = P + P1
540 NEXT T
545 PRINT
550 PRINT
900 END
```

Puntos de interés

- Se han previsto huecos en la numeración de las líneas para la inserción de módulos adicionales que se discutirán en el ejercicio propuesto al final del capítulo.
- La porción de programa encargada de ejecutar la ecuación del modelo es extraordinariamente sencilla.

27.6 Conclusión

En este capítulo se han expuesto la idea general de simulación y el concepto asociado a ella de **modelo.** Tras esbozar a grandes rasgos el proceso de elaboración de un modelo, se ha propuesto una aplicación práctica de dicho proceso. Los puntos más importantes son:

- Un modelo es una simplificación deliberada y ordenada de un sistema que tiene por objeto detectar las pautas esenciales de comportamiento de éste.
- Un modelo matemático es el que consta de ecuaciones que describen las pautas esenciales de comportamiento del sistema que simula.
- Una simulación por ordenador es la que se realiza por medio de un programa que contiene un modelo del sistema simulado.
- La simulación tiene aplicaciones numerosas y variadas.
- El proceso de modelado de un sistema es difícil y exige un enfoque sistemático.
- Un modelo bien diseñado constituye una representación válida de los rasgos esenciales del sistema, arroja alguna luz sobre su funcionamiento y es sencillo.

27 Ejercicio

1. Defina brevemente los siguientes términos o expresiones: simulación, modelo, modelo matemático, refinar un modelo.
2. El modelo matemático desarrollado en las secciones 27.4 y 27.5 sólo funciona para ciertos valores de los parámetros, las condiciones iniciales y los intervalos de tiempo; así, no funcionará si el intervalo de tiempo es cero.
   a) Bien por ensayo y error, bien por reflexión atenta sobre la ecuación del modelo, determine los intervalos de valores de cada una de las variables de entrada compatibles con el funcionamiento del modelo.

b) Vuelva a escribir el segmento de entrada del programa ejemplo 27.1 de forma tal que sean convalidadas todas las entradas. Si alguna de ellas está fuera del intervalo de validez, aparecerá un mensaje que instará al usuario a volver a introducirla.

3. *a*) Escriba la documentación para el programador y la guía para el usuario correspondientes al programa ejemplo 27.1.

b) Modifique el programa para que ofrezca al principio una serie de instrucciones dirigidas al usuario.

4. Escriba un módulo adicional para el programa ejemplo 27.1 que presente en pantalla o imprima una curva de nivel de población-tiempo.

5. Utilice el programa ejemplo 27.1 o una versión modificada del mismó para investigar algunas de las siguientes situaciones o todas ellas:

a) La población de cierto insecto en el campo se mantiene estable al nivel de 1.000 individuos. En condiciones ideales, esa población se triplicaría en un año (por tanto, $A = 2$). En el campo se acaba de plantar una especie que proporcionará al insecto 100 veces más alimento que la anterior. Suponiendo que el tamaño límite de la población es ahora de 100.000 individuos, investigue la evolución de la población a lo largo de los próximos 20 años.

b) Una manada de elefantes africanos tiene un tamaño de 100 individuos. Su tasa de crecimiento anual en condiciones ideales del 5 por 100 (por tanto, $A = 0{,}05$). La extensión de los cultivos ha reducido su territorio a una cuarta parte de la superficie habitual. Suponiendo que el tamaño límite de la manada es ahora 25 individuos, investigue la evolución de la población a lo largo de los próximos 30 años.

c) Accidentalmente se ha liberado una pareja de conejos en una región virgen para esa especie y capaz de soportar una población de 20.000 individuos. Teniendo en cuenta que en condiciones ideales una familia de conejos puede cuadruplicarse en un año (por tanto, $A = 3$), investigue la evolución de la población de conejos en esa región a lo largo de los 20 años siguientes.

d) La población humana de la tierra era de 3.600 millones de personas en 1970. La actual tasa de crecimiento (que no se ha frenado prácticamente nada) es del 2 por 100 anual. Haga una estimación del tamaño límite de la población humana (las opiniones de los expertos oscilan entre 1.000 y 70.0000 millones) e investigue su crecimiento a lo largo de los próximos 100 años. Verifique los resultados obtenidos desde 1970 hasta ahora comparándolos con las cifras reales de población en esos dos años.

e) Utilice las cifras proporcionadas por el censo para determinar el comportamiento de la población en un país determinado. Estime los parámetros del modelo y compare las cifras simuladas con las reales.

6. El modelo de población desarrollado en las secciones 27.3 y 27.4 puede ampliarse para que tenga en cuenta la extinción de una especie por actividades como la caza y la pesca. Si D es el número de individuos muertos por cazadores o pescadores por unidad de tiempo, la ecuación del modelo sería:

$$\Delta P = (A\left(1 - \frac{P}{L}\right)P - D)\,\Delta t.$$

Modifique el programa ejemplo 27.1 para incorporar este nuevo parámetro, que también debe ser convalidado, y utilícelo para investigar algunas de las situaciones siguientes, o todas ellas:

a) La población de cierta especie de pez permanece estabilizada en una zona de pesca en un millón de individuos. Se estima que, en condicio-

nes ideales, dicha población podría crecer a un ritmo del 10 por 100 anual. Investigue las consecuencias de pescar 1.000, 10.000 y 100.000 individuos al año. Determine el número máximo de capturas anuales que permitiría mantener una población estable y el tamaño de esa población estabilizada.

b) Obtenga datos suficientes para elaborar un modelo de la población de alguna especie amenazada, como la ballena azul. Utilícelo para investigar las consecuencias del ritmo actual de capturas y deduzca de los resultados algún procedimiento para garantizar la supervivencia de esa especie.

7. Son numerosas las situaciones que implican un elemento de incertidumbre o azar. Aunque la estadística permite determinar el comportamiento medio de tales sistemas, la simulación de una imagen mucho más completa de su funcionamiento. Algunas situaciones analizables de esta forma son la red telefónica, los cruces controlados por semáforos, el servicio postal, la facturación de un supermercado y los juegos.

El principio general en que se basa la simulación se llama **partición temporal** y consiste en la determinación, durante un breve período de tiempo, de las variaciones experimentadas por diversos valores numéricos del sistema por medio de números aleatorios. A continuación, esos valores se actualizan para el siguiente intervalo de tiempo. Por lo general, los intervalos elegidos son muy breves, con el fin de limitar el número de hechos que puedan ocurrir durante el transcurso de uno de ellos.

Daremos a continuación un ejemplo de este método, aplicado a la simulación de una central telefónica.

La central cuenta con cierto número de extensiones y recibe un número menor de líneas del exterior. Durante un intervalo de tiempo pequeño (10 segundos es un valor adecuado a este caso), existe cierta probabilidad de que se realice una llamada desde una de las extensiones a otra o a una línea exterior, y cierta probabilidad diferente de que se reciba una llamada desde el exterior. Durante el mismo período de tiempo considerado hay además cierta probabilidad de que, si alguna línea está ocupada durante el mismo, finalice esa ocupación. Se ignoran las probabilidades de que una llamada empiece y termine dentro del período estudiado.

El inicio de una llamada a una extensión se simula como sigue: durante cada intervalo de tiempo, se elige un número aleatorio comprendido entre 0 y 1; si el número es inferior a la probabilidad de que se produzca la llamada, se dice que la llamada se ha producido. En este caso, se elige otro número aleatorio para determinar el destino de la llamada; si dicho destino está comunicando, la llamada no tiene lugar.

La mejor forma de describir la situación es numerar las extensiones y las líneas exteriores y utilizar una matriz para indicar la situación de cada una. El valor 0 indica línea libre y un entero denota una llamada efectuada a o desde la extensión o la línea exterior a que corresponde ese número entero.

a) Complete el proceso de modelado y diseñe un programa que lo ponga en funcionamiento.

b) Utilice la misma técnica para simular otras situaciones mencionadas en la pregunta o inventadas por usted.

8. P. H. Leslie ha propuesto un método más complejo que el descrito en el texto para simular el tamaño de una población. En su formulación más sencilla, el modelo se refiere únicamente a las hembras, a las que divide en grupos de edad.

Por ejemplo, si la población hembra de una determinada especie se divide en tres grupos —de 0 a 9 años, de 10 a 19 y de 20 años en

adelante— el número de ejemplares de cada uno de esos grupos puede expresarse en forma de vector columna:

$$N = \begin{bmatrix} 300 \\ 250 \\ 150 \end{bmatrix}$$ que indica que hay 300 hembras de 0 a 9 años, etc.

El cambio experimentado por la población en un período de tiempo (10 años en este caso) se describe mediante una matriz como esta:

$$M = \begin{bmatrix} 0 & 5 & 2 \\ 0{,}8 & 0 & 0 \\ 0 & 0{,}6 & 0{,}4 \end{bmatrix}$$

Las cifras de la parte superior corresponden al número medio de crías hembras nacidas de cada hembra y en cada grupo de edad durante el período considerado. Así, la cifra 5 significa que, por término medio, cabe esperar que cada una de las hembras pertenecientes al grupo de edad de 10 a 19 años tenga cinco hijas en un período de 10 años.

Las cifras situadas por debajo de la diagonal representan las probabilidades de que un individuo sobreviva hasta pasar al siguiente grupo de edad. Por ejemplo, 0,8 es la probabilidad de que un individuo perteneciente al grupo de 0 a 9 años pase a formar parte del grupo de 10 a 19 años.

La cifra del ángulo inferior derecho representa la probabilidad de que un individuo del grupo de edad superior sobreviva hasta el siguiente período de tiempo. En este caso, hay un 40 por 100 de probabilidades de que un cierto individuo del grupo de edad superior se mantenga vivo diez años más.

La multiplicación de la matriz por el vector de población da lugar al vector de población correspondiente al período de tiempo siguiente. Es decir:

$$\begin{bmatrix} 0 & 5 & 2 \\ 0{,}8 & 0 & 0 \\ 0 & 0{,}6 & 0{,}4 \end{bmatrix} \times \begin{bmatrix} 300 \\ 250 \\ 150 \end{bmatrix} = \begin{bmatrix} 1.550 \\ 240 \\ 210 \end{bmatrix}$$

Como se ve, la nueva población tiene 1.550 hembras de 0 a 9 años, 240 de 10 a 19 y 210 de 20 o más.

a) Escriba un programa que lleve a la práctica el modelo de población de Leslie tal como acaba de exponerse. Utilice los datos del ejemplo o los procedentes de alguna población real, como la de ballenas azules, que se indican a continuación.

La matriz de Leslie para la población de ballenas azules sobre un período de 2 años es, según estimaciones:

$$W = \begin{bmatrix} 0 & 0 & 0{,}19 & 0{,}44 & 0{,}50 & 0{,}50 & 0{,}45 \\ 0{,}87 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}87 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}87 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}87 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}87 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0{,}87 & 0{,}80 \end{bmatrix}$$

b) Modifique el modelo para tener en cuenta la presión ambiental, según se ha hecho en el texto. Para ello se reduce cada elemento de la

matriz con arreglo a un factor de la población total. Lo más sencillo es colocar estos factores en una segunda matriz, que llamaremos F, y realizar el cálculo

$$(M - FP) \times N$$

donde P es la población total en cada período de tiempo.

La matriz correspondiente a los factores ambientales sería en el ejemplo que nos ocupa:

$$F = \begin{bmatrix} 0 & 0{,}001 & 0{,}002 \\ 0{,}0001 & 0 & 0 \\ 0 & 0{,}0001 & 0{,}0002 \end{bmatrix}$$

El cálculo de la nueva población se efectúa como sigue:

$$P = 300 + 250 + 150 = 700$$

(suma de los elementos de N)

$$FP = \begin{bmatrix} 0 & 0{,}7 & 1{,}4 \\ 0{,}07 & 0 & 0 \\ 0 & 0{,}07 & 0{,}14 \end{bmatrix}$$

$$M - FP = \begin{bmatrix} 0 & 4{,}7 & 0{,}6 \\ 0{,}73 & 0 & 0 \\ 0 & 0{,}53 & 0{,}26 \end{bmatrix}$$

$$(M - FP) \times N = \begin{bmatrix} 0 & 4{,}9 & 0{,}6 \\ 0{,}73 & 0 & 0 \\ 0 & 0{,}53 & 0{,}26 \end{bmatrix} \times \begin{bmatrix} 300 \\ 250 \\ 150 \end{bmatrix} = \begin{bmatrix} 1.265 \\ 219 \\ 172 \end{bmatrix}$$

Como se ve, las nuevas cifras de población son inferiores a las alcanzadas por la misma población sin restricciones ambientales.

c) Modifique el modelo para incluir en cada grupo de edad el número de individuos capturados por prácticas de caza o pesca durante cada intervalo de tiempo. Investigue los efectos sobre la población de diferentes pautas de captura.


RESET
RETURN

28
Análisis sintáctico

En informática, el análisis sintáctico forma parte del proceso de traducción de un programa de un lenguaje a otro, por lo general de un lenguaje de alto nivel, como el Basic, a otro nivel inferior, como el lenguaje máquina o el ensamblador. Los programas encargados de realizar este proceso de traducción se llaman compiladores o intérpretes y forman parte del soporte lógico que hace de un ordenador un instrumento útil.

Por tanto, este capítulo se referirá a las estructuras del ordenador y del lenguaje de programación más que a una aplicación concreta de la informática. Discutiremos técnicas de análisis sintáctico y aplicaremos dos de ellas a otros tantos casos concretos de análisis. También diseñaremos programas para llevar a la práctica esas dos técnicas. Terminará el capítulo con una breve discusión en torno a otro enfoque del análisis sintáctico.

Utilizaremos material de capítulos anteriores, y sobre todo del 18, dedicado al estudio de la estructura de datos llamada pila. Sin embargo, los restantes capítulos no tendrán mucha relación con éste.

28.1 Enfoques del análisis sintáctico

Se llama análisis sintáctico al reconocimiento de la estructura de un programa, por lo general escrito en un lenguaje de alto nivel, como el Basic. Los programas se organizan en sentencias que incluyen

palabras de instrucción, expresiones, nombres de variables, números, etcétera, elementos todos ellos que deben ser identificados hasta el último carácter.

Hay dos razones para proceder al análisis sintáctico: una es localizar e identificar los posibles errores cometidos en la redacción del programa; la otra es generar un código en lenguaje ensamblador o en lenguaje máquina que sea equivalente al programa escrito en lenguaje de alto nivel.

Los enfoques del análisis sintáctico varían y, en cierta medida, dependen del lenguaje en el que está escrito el programa compilador o intérprete. Hay un enfoque **descendente** que empieza por considerar la estructura general del programa y a continuación va descomponiendo éste en subestructuras cada vez menores hasta llegar a los componentes mínimos. El inconveniente de este procedimiento es que el compilador debe tener subprogramas **recurrentes,** incompatibles con la mayor parte de las versiones del Basic.

El otro enfoque, **ascendente,** identifica por separado cada uno de los componentes del programa y determina a partir de ellos la estructura general. También se llama método **tabular,** porque necesita de varias tablas para identificar los diversos componentes. Es el método que seguiremos aquí, ya que es el único compatible con la versión de referencia del Basic, aunque de todas formas haremos una breve exposición del otro procedimiento en la sección 28.7.

28.2 Areas del análisis sintáctico

Naturalmente, el análisis exhaustivo de la sintaxis de un programa exige el de cada una de sus partes, pero hay dos casos en que, por su importancia, la atención se concentra todavía más: el reconocimiento de las series de caracteres alfanuméricos que constituyen números válidos y el reconocimiento de expresiones aritméticas. Estudiaremos estas dos áreas de interés en las cuatro próximas secciones, y diseñaremos y escribiremos sendos programas para llevar a la práctica los resultados. En todos los casos seguiremos un enfoque tabular orientado de abajo hacia arriba.

28.3 Reconocimiento de números: tabla de estados

Prácticamente todos los lenguajes de programación tienen previsto el uso de números, que casi siempre pueden ser de varios tipos:

enteros, fracciones y números escritos con coma flotante. La mayoría de los lenguajes de alto nivel trabajan con numeración de base decimal, pero de todas formas ésta no es una norma obligatoria. También los detalles sobre lo que es un número válido cambian de unos lenguajes a otros.

Reconocer un número consiste en examinarlo carácter por carácter para determinar si es correcto o no. Este proceso ocurre por lo general simultáneamente a otro que reduce el número a la base binaria en que trabaja el ordenador.

Una forma de reconocer números es recurrir a una **tabla de estados,** consistente en una serie de filas y columnas numeradas que corresponden a los caracteres válidos que pueden formar parte del número. El cuerpo de la tabla está formado por una serie de números que indican la fila a la que se transfiere el control cuando se localiza el carácter que encabeza la columna correspondiente. El análisis empieza en el estado 1.

Así, un **entero con signo** puede definirse como sigue:

Un entero con signo consta de	un	signo opcional + o − seguido de
	una	o más cifras decimales seguidas de
	un	carácter terminal ∇.

La tabla de estado correspondiente a la definición sería:

Estado	+	−	*Cifra*	*Otro carácter*	∇
1	2	2	3	error 1	error 2
2	error 3	error 3	3	error 1	error 2
3	error	error 3	3	error 1	salida

Observe que hay varias situaciones de error y una sola salida correcta.

El análisis del número −174∇ se desarrolla como sigue:

Estado anterior	*Carácter*	*Nuevo estado*
1	—	2
2	1 (cifra)	3
3	7 (cifra)	3
3	4 (cifra)	3
3	∇	salida

El número ha sido considerado un entero con signo correcto. Veamos ahora otro ejemplo: el número 41 + 3∇:

Estado anterior	*Carácter*	*Nuevo estado*
1	4	3
3	1	3
3	+	error 3

El análisis termina en este punto. Error 3 puede interpretarse como la presencia de un signo dentro de un entero. Las otras dos situaciones de error podrían interpretarse de forma similar.

28.4

Programa ejemplo 28.1

El objetivo de este programa es aplicar en la práctica el proceso de reconocimiento de un entero con signo por medio de una tabla de estado. El método se basa en lo expuesto en la sección anterior.

Diseño del programa

La estructura general del programa es la siguiente:

Introducir un entero con signo.
Analizar la sintaxis del mismo.
Presentar a la salida un mensaje que diga que la sintaxis es correcta o que establezca el error cometido.

Lo mejor es utilizar una variante de la tabla de estado en la que las condiciones de error y salida y las columnas quedan identificadas por números:

Carácter	+	−	*Cifra*	*Otro carácter*	∇
Tipo de carácter	1	2	3	4	5
estado					
1	2	2	3	4	5
2	6	6	3	4	5
3	6	6	3	4	7

Los estados 4, 5 y 6 son situaciones de error y el 7 es la salida. Cualquiera de dichos estados hace que el programa pase de la fase de análisis a la de salida, durante la que produce un mensaje numerado. La tabla de estado se almacena en una matriz.

Con ayuda de la nueva tabla de estado, el programa puede ya descomponerse de forma más detallada:

Cargar la matriz correspondiente a la tabla de estado y los mensajes.

Introducir un entero con signo en forma de serie de caracteres alfanuméricos.

Añadir el carácter terminal.

Hacer estado = 1 y puntero del carácter = 1.

Repetir el proceso:

Determinar el tipo de carácter en curso.
Usar la tabla para actualizar el estado.
Incrementar el puntero de caracteres.

Hasta que el estado sea > 3.

El número de estado es el mensaje que se lleva a la salida.

La determinación del tipo del carácter en curso es mejor hacerla en un subprograma. El parámetro transferido a dicho subprograma es el carácter en curso del entero, y el parámetro devuelto es el tipo de ese carácter.

Variables

T(3, 5)	Tabla de estados.
M$(4)	Mensajes que indican el resultado del análisis.
E	Estado en curso.
E$	Entero con signo.
S	Puntero orientado hacia el carácter en curso del entero.
C$	Carácter en curso y parámetro del subprograma.
Y	Tipo de carácter y parámetro del subprograma.
I, J, K	Contadores de bucle.

Diagrama de flujo

Véase la figura 28.1.

Programa

```
100 REM PROGRAMA EJEMPLO 28.1
105 REM ANALISIS DE UNA TABLA DE ESTADO DE
110 REM ENTEROS CON SIGNO
115 REM
```

Inicialización

```
200 REM INICIALIZACION
205 DIM T(3, 5), M$(4)
210 REM CARGA DE LA TABLA DE ESTADO
```

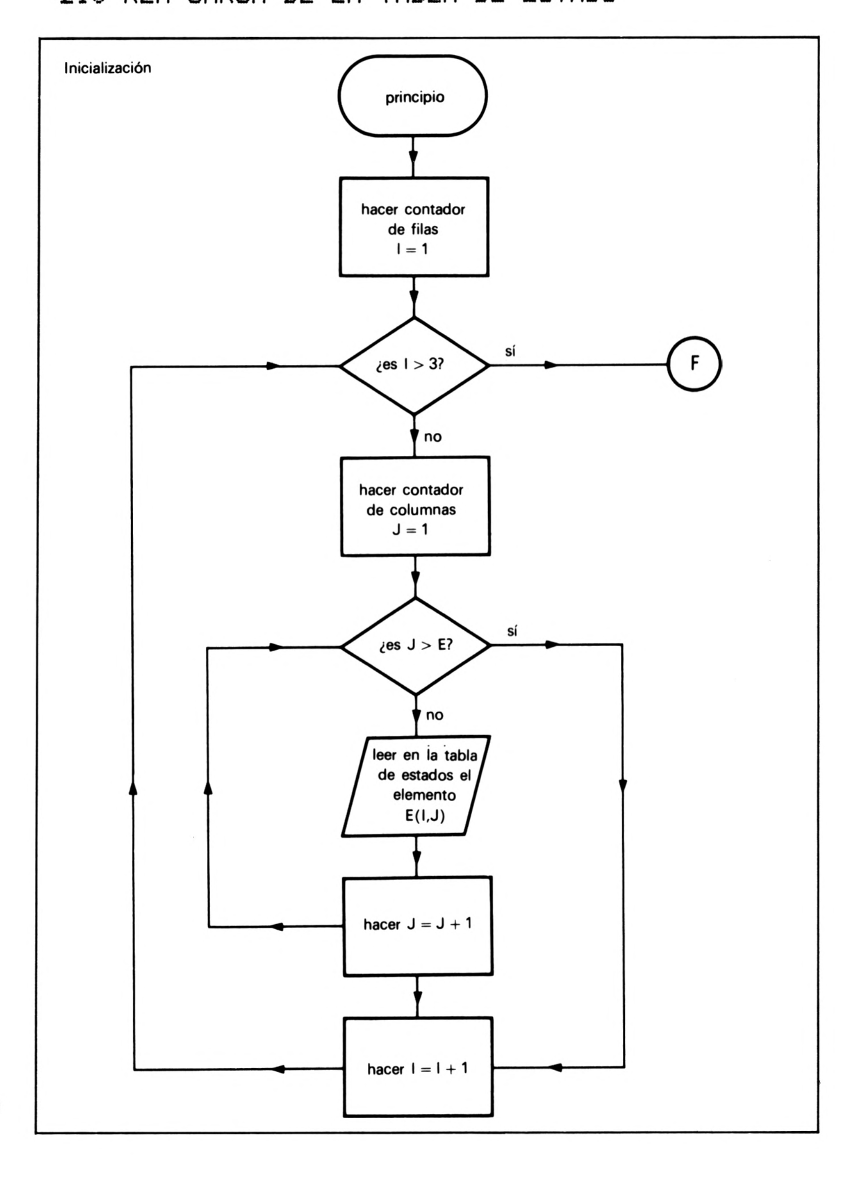
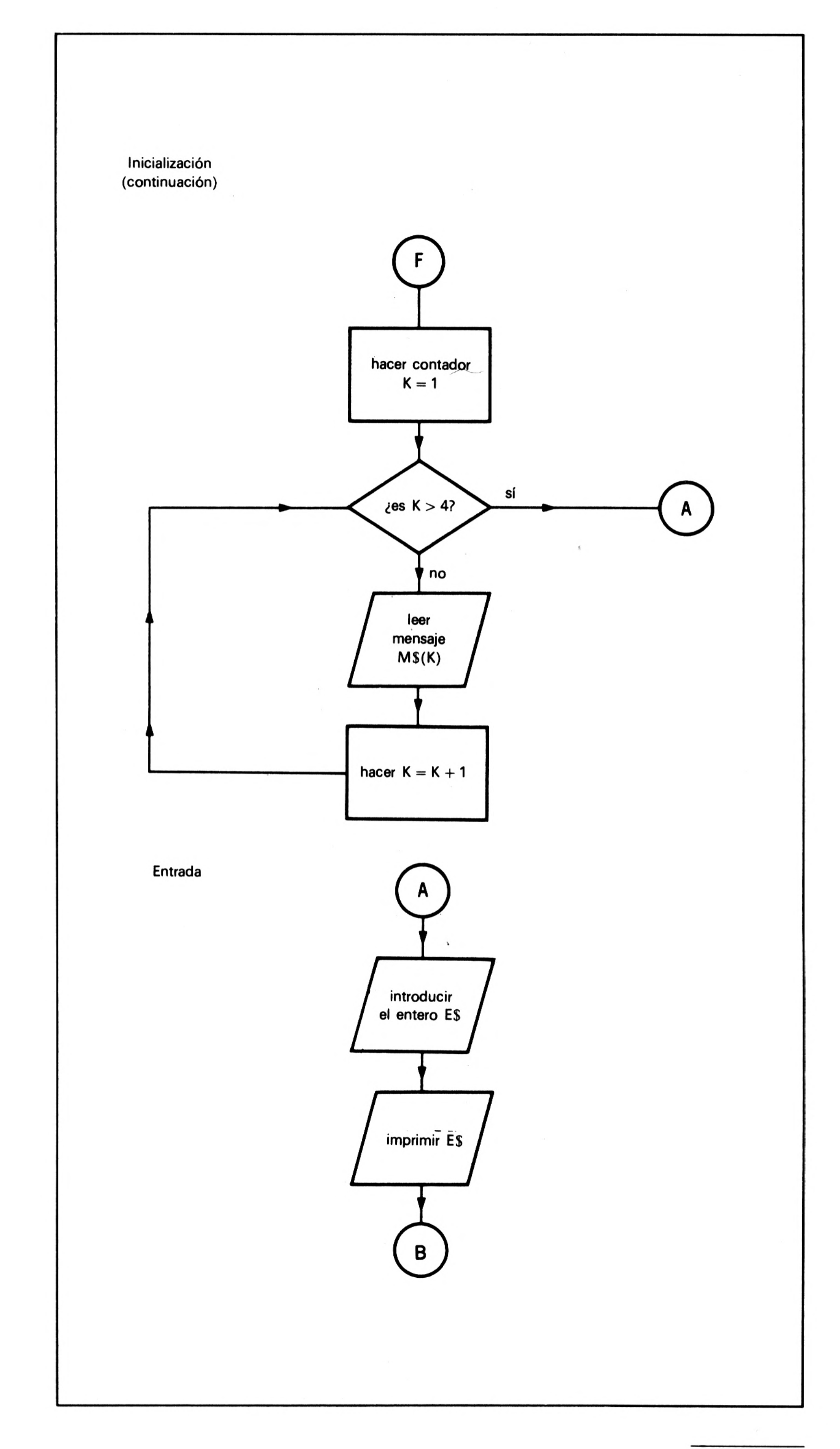
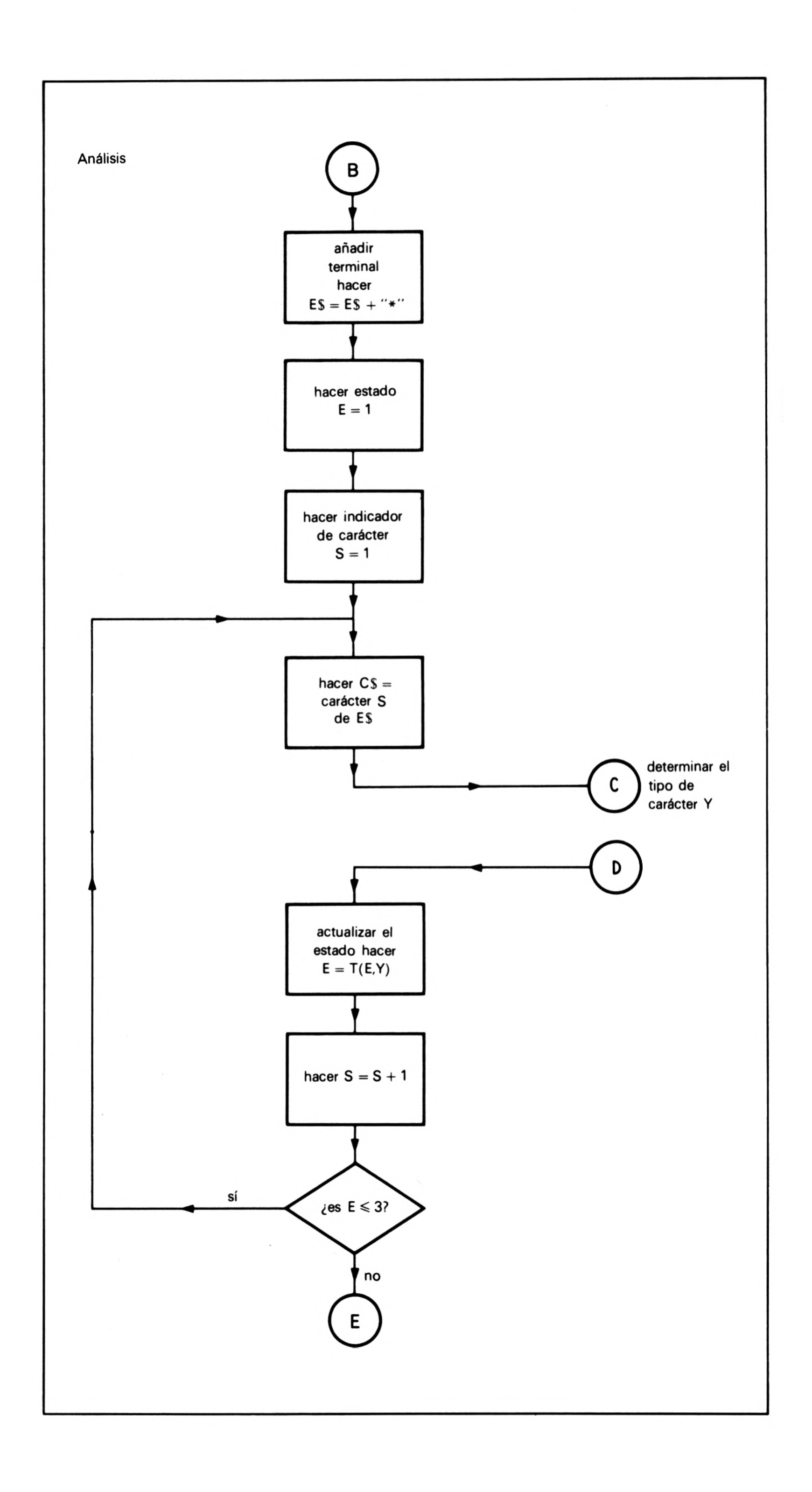
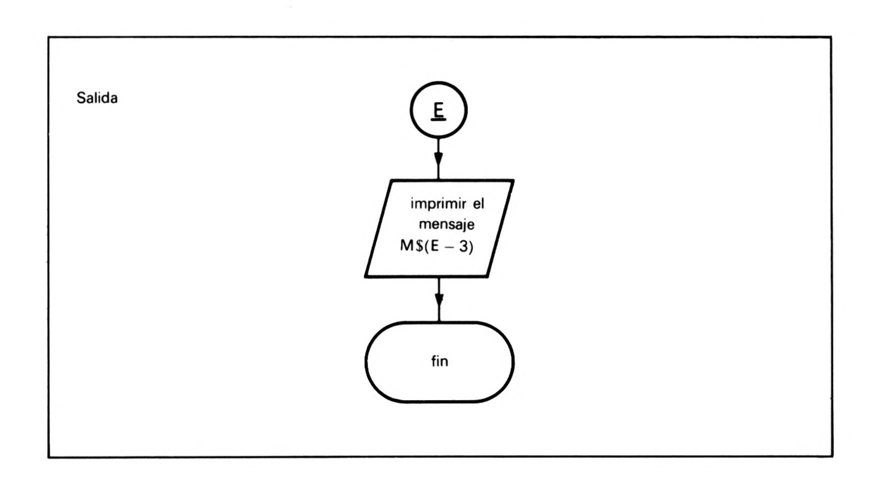


Figura 28.1
Diagrama de flujo del programa ejemplo 28.1


Inicialización
(continuación)
F
hacer contador
K = 1
¿es K > 4?
sí
A
no
leer
mensaje
M$(K)
hacer K = K + 1
Entrada
A
introducir
el entero E$
imprimir E$
B



Análisis
B
añadir
terminal
hacer
E$ = E$ + "*"
hacer estado
E = 1
hacer indicador
de carácter
S = 1
hacer C$ =
carácter S
de E$
C
determinar el
tipo de
carácter Y
D
actualizar el
estado hacer
E = T(E,Y)
hacer S = S + 1
¿es E ≤ 3?
sí
no
E



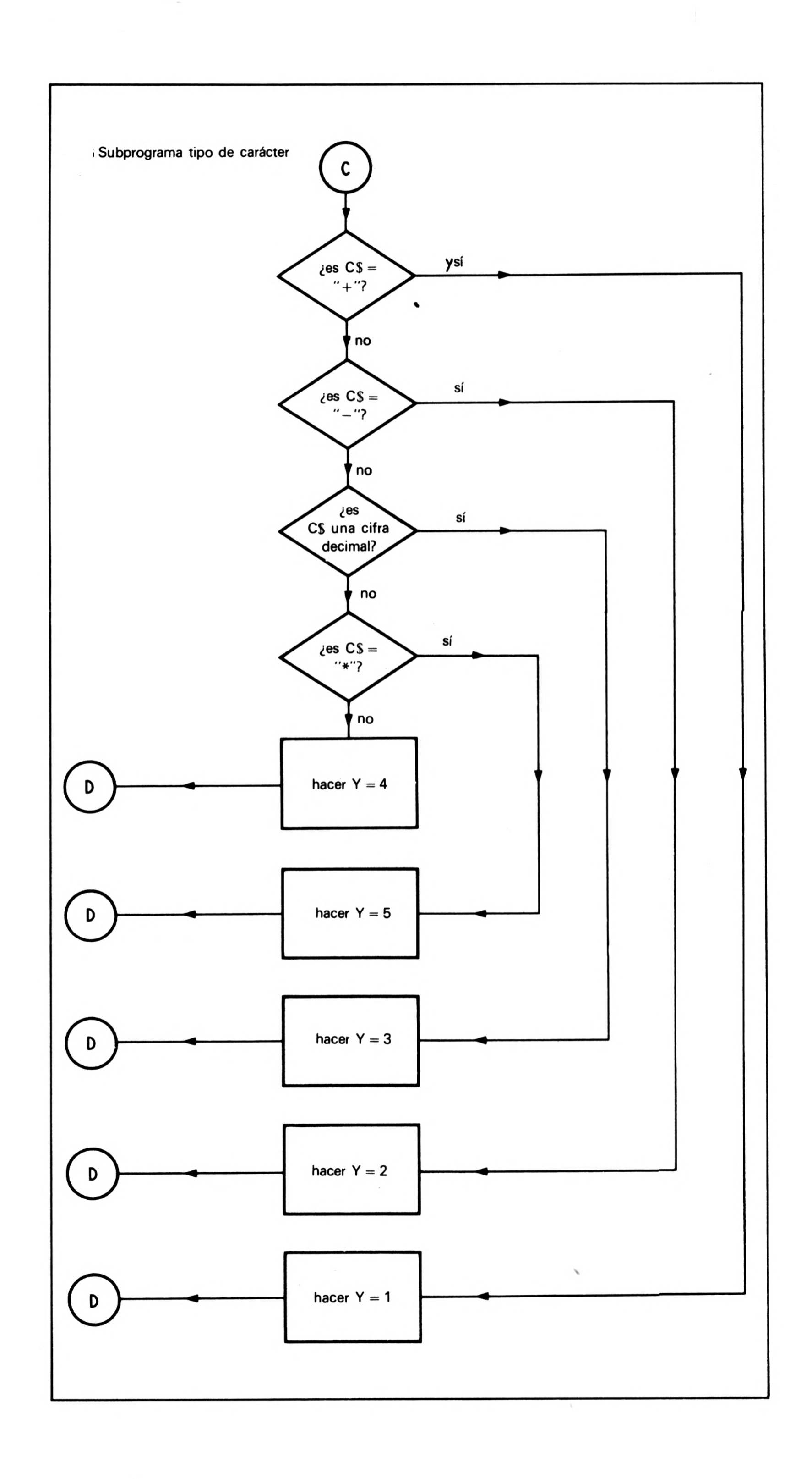

```
215 FOR I = 1 TO 3
220 FOR J = 1 TO 5
225 READ T(I, J)
230 NEXT J
235 NEXT I
240 DATA 2, 2, 3, 4, 5
245 DATA 6, 6, 3, 4, 5
250 DATA 6, 6, 3, 4, 7
255 REM MENSAJES DE CARGA
260 FOR K = 1 TO 4
265 READ M$(K)
270 NEXT K
275 DATA "CARACTER NO VALIDO"
280 DATA "NUMERO INCOMPLETO"
285 DATA "ENTERO CON SIGNO EN SU INTERIOR"
290 DATA "ENTERO CORRECTO"
295 REM
```

Entrada

```
300 REM INTRODUCCION
305 INPUT E$
310 PRINT E$,
315 REM
```

Análisis

```
400 REM ANALISIS
405 LET E$ = E$ + "*"
410 LET E = 1
415 LET S = 1
420 REM
500 REM BUCLE DE CONTROL
```


Subprograma tipo de carácter
C
¿es C$ =
"+"?
ysí
no
¿es C$ =
"−"?
sí
no
¿es
C$ una cifra
decimal?
sí
no
¿es C$ =
"*"?
sí
no
hacer Y = 4
D
hacer Y = 5
D
hacer Y = 3
D
hacer Y = 2
D
hacer Y = 1
D

```
505 LET C$ = MID$(E$, S, 1)
510 GOSUB 700: REM DETERMINA EL TIPO DEL
                          CARACTER
515 LET E = T(E, Y)
520 LET S = S + 1
525 IF E <= 3 THEN 500
530 REM
```

Salida

```
600 REM SALIDA
605 PRINT M$(E - 3)
610 STOP
```

Subprograma "tipo de carácter"

```
700 REM SUBPROGRAMA DE TIPO DEL CARACTER
705 REM PARAMETROS
710 REM C$: CARACTER
715 REM Y: ESCRITURA DEL CARACTER
720 IF C$ = "+" THEN 780
725 IF C$ = "-" THEN 770
730 IF ASC(C$) >= 48 AND ASC(C$) <= 57 THEN 760
735 IF C$ = "*" THEN 750
740 LET Y = 4
745 RETURN
750 LET Y = 5
755 RETURN
760 LET Y = 3
765 RETURN
770 LET Y = 2
775 RETURN
780 LET Y = 1
785 RETURN
999 END
```

Puntos de interés

- En relación con la tarea que realiza, el programa es bastante largo.
- Observe que los estados 4 a 7 corresponden a los mensajes 1 a 4. El almacenamiento de los mensajes de error en una matriz tal como se ha hecho aquí es práctica habitual en los programas de análisis sintáctico.
- El subprograma utiliza una cadena de saltos para determinar el tipo de carácter. Esta secuencia de condiciones podría escribirse de forma mucho más concisa haciendo uso de la construcción **IF ... THEN ... ELSE.**
- El carácter terminal usado es *.

28.5

Sintaxis de expresiones aritméticas: tabla de prioridades

La mayor parte de los lenguajes de programación de alto nivel permiten introducir expresiones aritméticas en la notación habitual. Durante la compilación o interpretación del programa, se verifica la sintaxis de tales expresiones y se generan los correspondientes códigos de máquina.

Esta sección se centrará en la generación del código mediante un proceso de transformación de una expresión escrita en notación aritmética normal en otra escrita en **notación polaca inversa,** utilizando para ello una **tabla de prioridades** (en la pregunta 8 del ejercicio 21 se esbozó un procedimiento de transformación de la notación polaca inversa).

La notación polaca inversa se caracteriza porque en ella los signos de operación se escriben tras los números o letras a que hacen referencia **(operandos).** Vea en la pregunta 4 del ejercicio 18 algunos ejemplos de esta notación. A aquéllos añadiremos este otro:

Notación normal	*Notación polaca inversa*
$(A + B) * 2$	$AB + 2 *$

Observe que la notación polaca inversa no necesita paréntesis. La razón de esta transformación es que el código de máquina generado por la mayor parte de los compiladores se parece mucho a la notación polaca inversa de una expresión.

La tabla de precedencia está relacionada con el orden de prioridad de las diversas operaciones aritméticas, que es el siguiente:

Primero la multiplicación y la división.
Después la adición y la substracción.
Las subexpresiones contenidas entre paréntesis se efectúan antes que todas las demás, y en el orden arriba indicado.

Una tabla de prioridades no es sino una plasmación formalizada de estas normas, y determina el orden en que han de colocarse las operaciones en notación polaca inversa.

La transformación a notación inversa mediante una tabla de precedencia exige también el uso de una **pila** (véase capítulo 18). En líneas generales, el proceso es el siguiente:

Examinar la expresión carácter por carácter.
Si el carácter es un operando, extraerlo.
En caso contrario (si es un signo de operación), colocarlo en una **pila de operación.**

Comparar la operación en curso (situada en la cima de la pila) con la anterior (situada tras la cima) utilizando para ello la siguiente tabla de prioridades:

Operación anterior	*Operación en curso*					
	+	–	*	/	(	)
+	1	1	3	3	3	1
–	1	1	3	3	3	1
*	1	1	1	1	3	1
/	1	1	1	1	3	1
(	3	3	3	3	3	2
)	4	4	4	4	4	4

Los enteros que forman el cuerpo de la tabla se interpretan como sigue:

1: La operación anterior tiene prioridad. Mostrarla y retirarla de la pila. Repetir la comparación con la nueva operación anterior.
2: Paréntesis emparejados. Extraerlos de la pila.
3: La operación en curso tiene prioridad. No hacer nada.
4: Error: paréntesis mal emparejados.

Este proceso exige que siempre haya al menos dos operaciones en la pila, lo que se consigue encerrando la expresión entre paréntesis al principio, y no estableciendo ninguna comparación si la pila llega a tener menos de dos elementos. Los dos paréntesis adicionales se eliminan de la pila al final del proceso. Veamos, por ejemplo, el análisis de la expresión

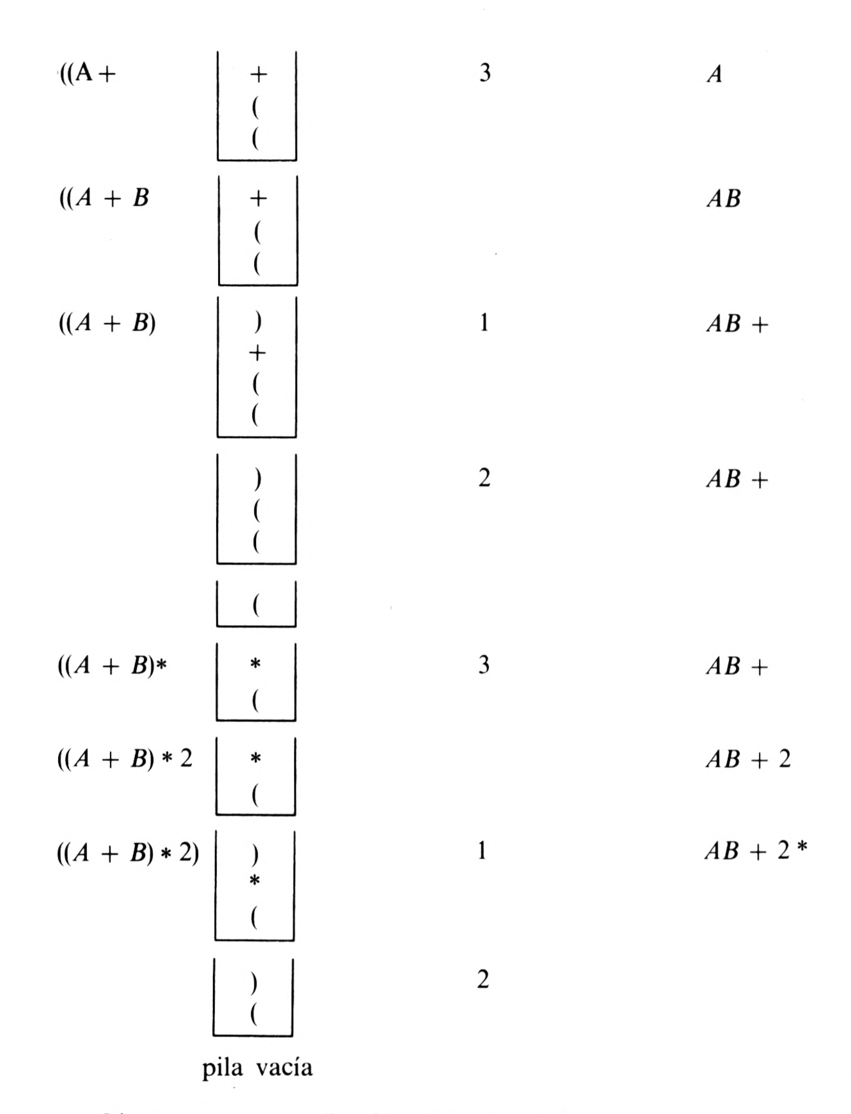
$(A + B) * 2$

Primero se encierra entre paréntesis:

$((A + B) * 2)$

Las fases de la transformación a notación polaca inversa son:

Entrada	*Pila*	*Entero de la tabla de prioridades*	*Salida*
(	(		
((	((	3	
((A	((		*A*



Observe que esta aplicación de la pila obliga a acceder al elemento situado por debajo de la cima, llevarlo a la salida y eliminarlo de la pila. No es una operación normal, pero es fácil de realizar en términos de las de introducir y extraer.

28.6

Programa ejemplo 28.2

El objetivo de este programa es transformar una expresión aritmética normal a la misma escrita en notación polaca inversa, utilizando

para ello el método que acabamos de describir. Por razones de sencillez, supondremos que la expresión contiene únicamente números de una sola cifra y variables de una sola letra; en otras palabras: que los operandos constan de un solo carácter. El programa no tiene rutina de convalidación: todo lo que no sea un símbolo de operación se supone que es un operando.

Diseño del programa

La estructura general del programa es la siguiente:

Cargar una lista de operaciones en la tabla de prioridad.

Inicializar el puntero de la pila para señalar una pila vacía.

Introducir una expresión.

Encerrarla entre paréntesis.

Analizar la sintaxis de la expresión y llevar a la salida su equivalente en notación polaca inversa.

Como la fase de análisis sintáctico ya ha sido analizada detalladamente en la sección anterior, la única fase del diseño que queda por especificar es la que se refiere a la representación y el almacenamiento de los datos.

Lo más fácil es representar las operaciones por los enteros correspondientes a sus posiciones en las filas y columnas de la tabla de prioridad (así, + = 1, − = 2, etc.). La pila está formada por tales enteros. Se introduce también una matriz de los símbolos de operaciones correspondientes a los enteros, que utilizan como índices.

Las operaciones de introducción y extracción se realizan mediante subprogramas. Al contrario que las rutinas escritas en el capítulo 18, esto permite que la pila crezca hacia abajo dentro de la matriz. El puntero señala la cima de la estructura de datos en lugar de la siguiente posición vacía y una pila vacía queda representada por un indicador de valor cero.

Teniendo en cuenta todas las consideraciones que acabamos de hacer, pasaremos a redactar un algoritmo para realizar el análisis sintáctico y la conversión a notación polaca inversa:

Para cada uno de los caracteres de la expresión, repetir el proceso:

```
Si el carácter es un operando,
    llevarlo a la salida;
en caso contrario, apilar el código de la operación;

    repetir el proceso:
        si hay al menos dos operaciones en la pila,
```

comparar la prioridad de la operación en curso con la de la anterior;

si el código de prioridad = 1, extraer la operación en curso, extraer la anterior y llevarla a la salida y apilar la operación en curso;
si el código = 2, extraer la operación en curso y la anterior;
si el código = 3, no hacer nada;
si el código = 4, presentar un mensaje de error,

hasta que el código de prioridad no sea igual a 1.

Este algoritmo tiene ya detalle suficiente como para escribir el programa a partir de él.

Variables

P(6, 6)	Tabla de prioridad.
S(20)	Pila de operaciones.
R$(6)	Matriz de símbolos de operaciones.
E$	Expresión.
K, L, M	Contadores de bucle.
I	Indicador de la pila.
A	Entero apilado o no apilado (parámetro del subprograma).
F	Indicador de resultado de los subprogramas de introducción y extracción de la pila.
C$	Carácter en curso de la expresión (parámetro del subprograma).
D	Entero que representa la operación en curso (parámetro del subprograma).

Diagrama de flujo

Véase la figura 28.2.

Programa

```
100 REM PROGRAMA EJEMPLO 28.2
105 REM CONVERSION A NOTACION POLACA INVERSA
110 REM
```

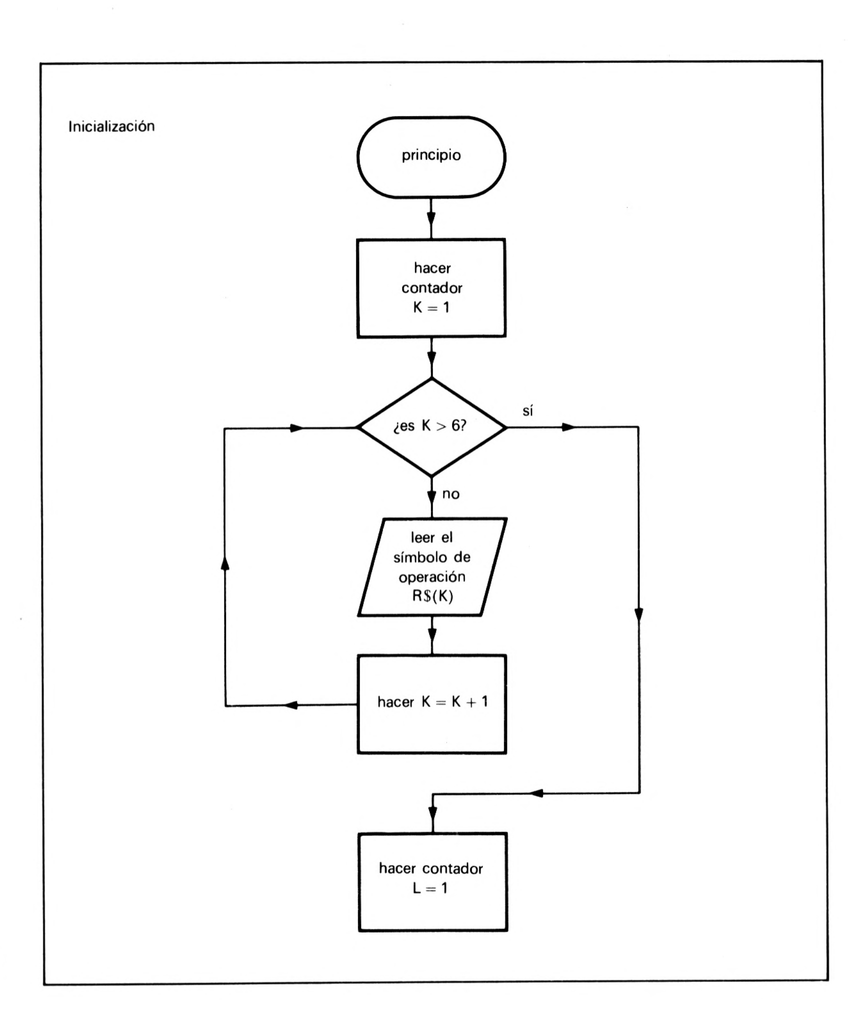
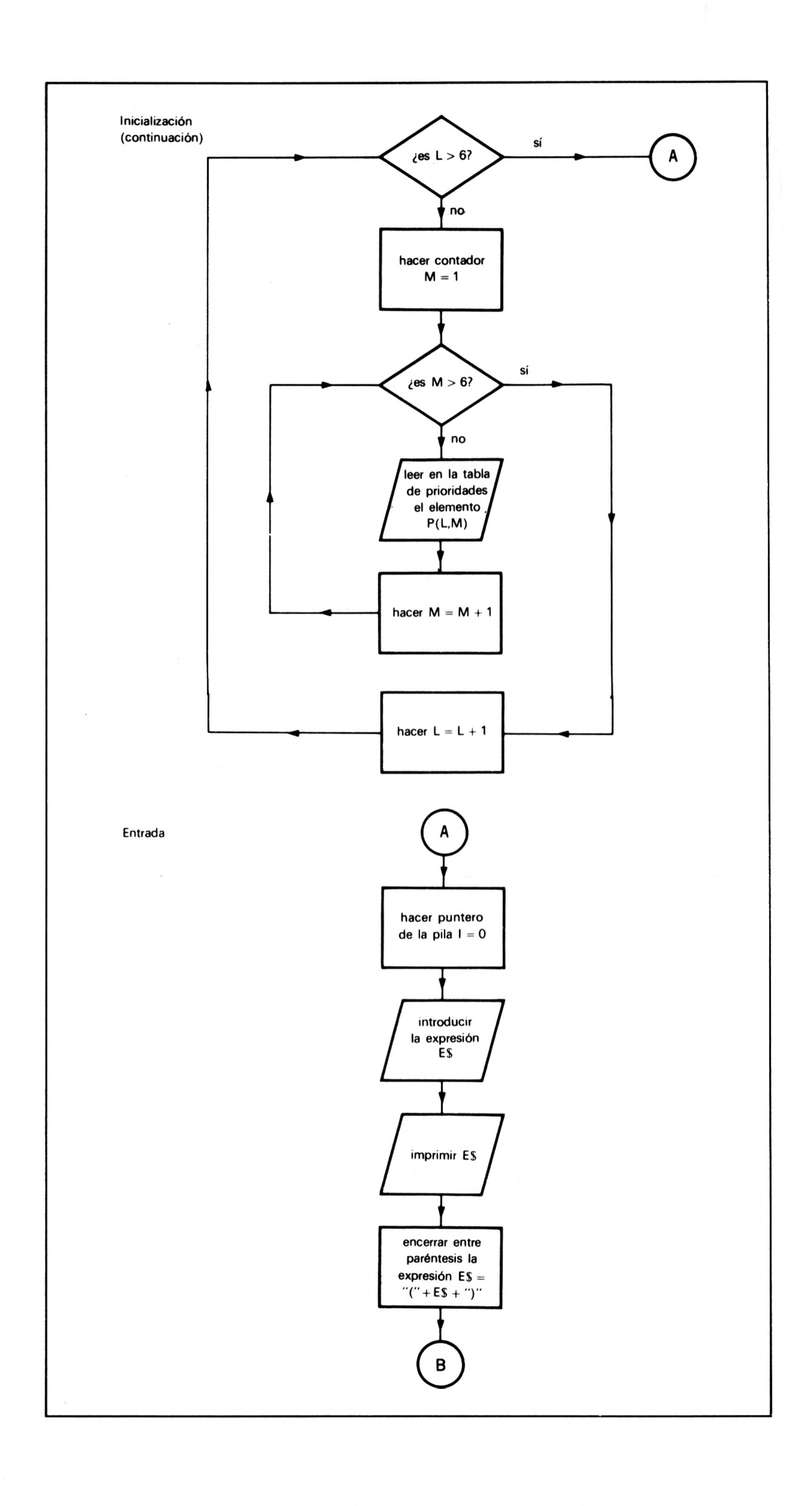


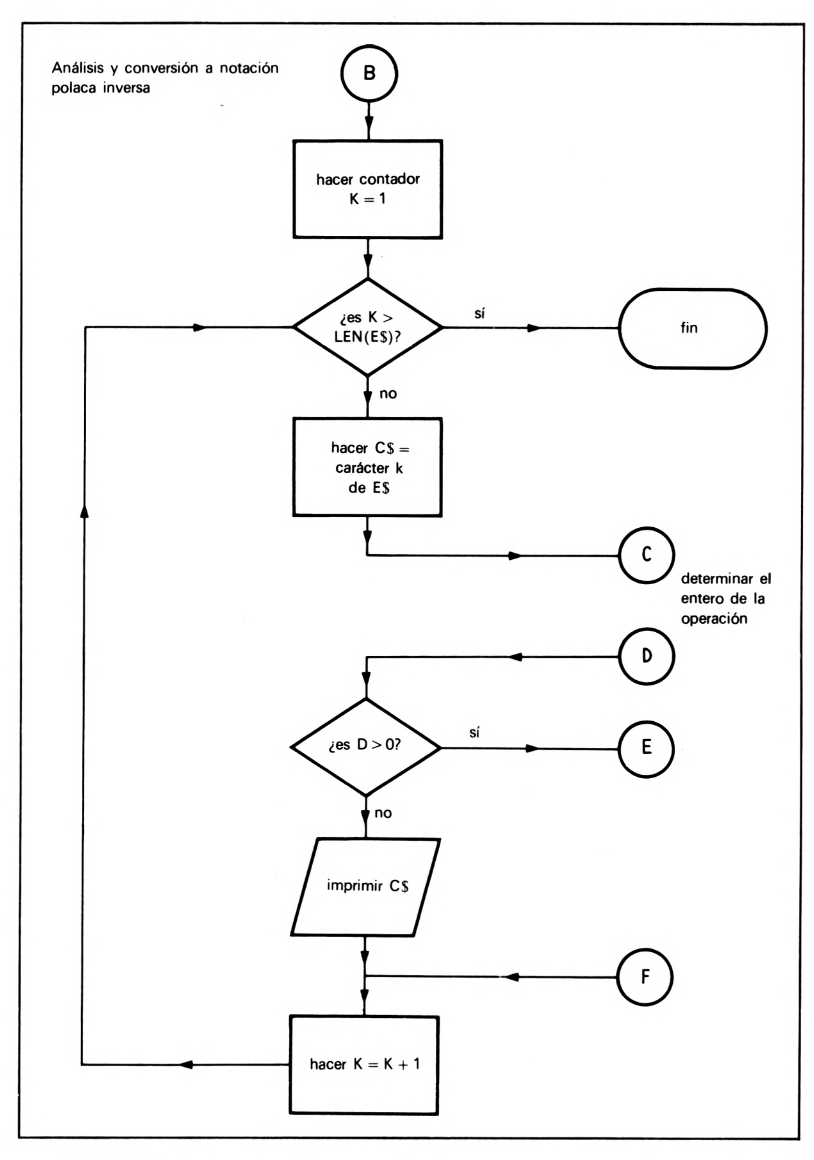
Figura 28.2
Diagrama de flujo del programa ejemplo 28.2

Inicialización

```
200 REM INICIALIZACION
205 REM CARGA DE LA MATRIZ
210 FOR K = 1 TO 6
215 READ R$(K)
220 NEXT K
225 DATA "+", "-", "*", "/", "(", ")"
230 REM CARGA DE LA TABLA ANTERIOR
235 FOR L = 1 TO 6
240 FOR M = 1 TO 6
245 READ P(L, M)
250 NEXT M
255 NEXT L
260 DATA 1, 1, 3, 3, 3, 1
265 DATA 1, 1, 3, 3, 3, 1
270 DATA 1, 1, 1, 1, 3, 1
```



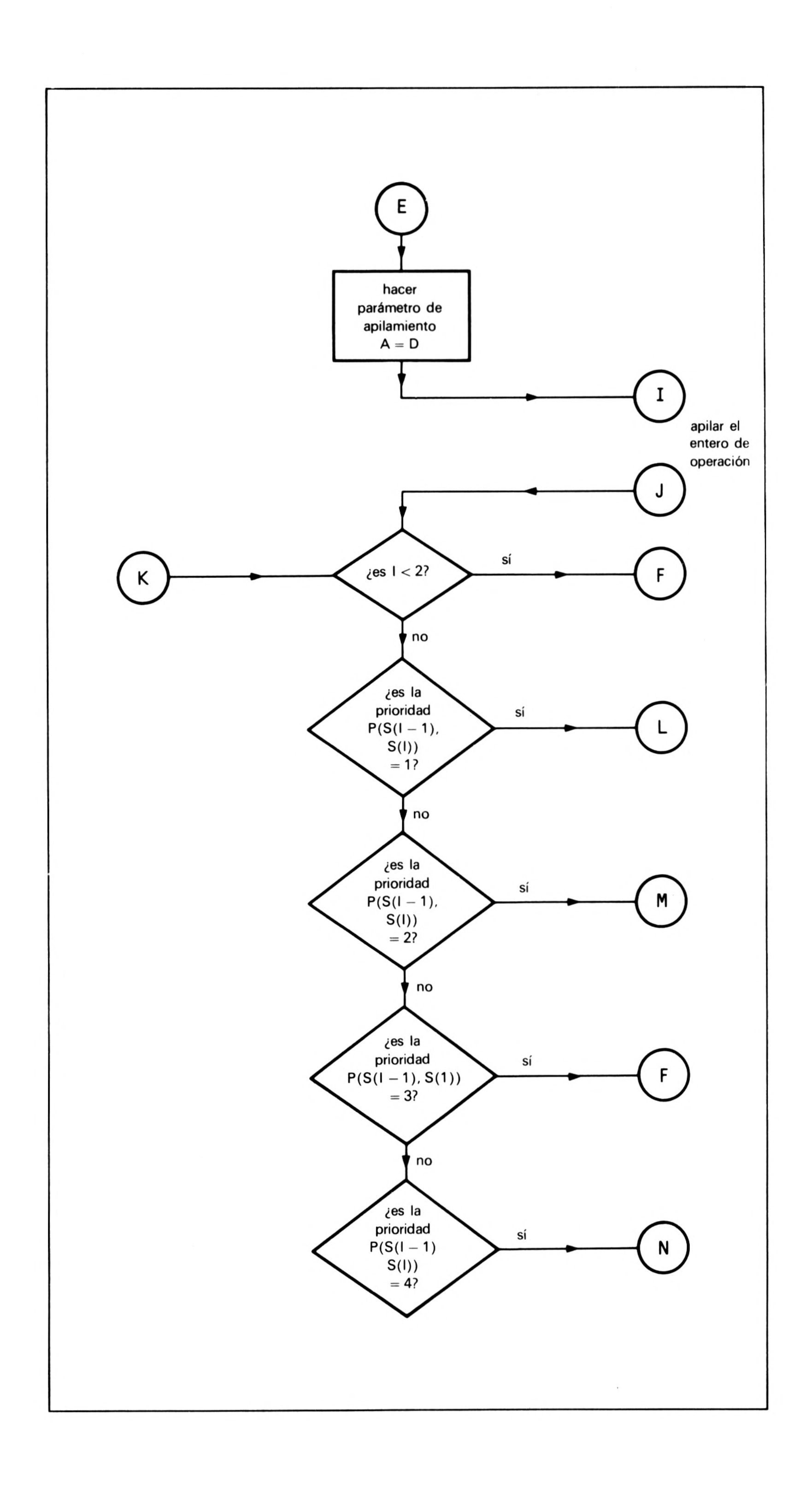
Inicialización
(continuación)
¿es L > 6?
sí
A
no
hacer contador
M = 1
¿es M > 6?
sí
no
leer en la tabla
de prioridades
el elemento
P(L,M)
hacer M = M + 1
hacer L = L + 1
Entrada
A
hacer puntero
de la pila I = 0
introducir
la expresión
E$
imprimir E$
encerrar entre
paréntesis la
expresión E$ =
"(" + E$ + ")"
B




```
275 DATA 1, 1, 1, 1, 3, 1
280 DATA 3, 3, 3, 3, 3, 2
285 DATA 4, 4, 4, 4, 4, 4
290 REM
```

Entrada

```
300 REM ENTRADA
305 LET I = 0
310 INPUT E$
```


E
hacer
parámetro de
apilamiento
A = D
I
apilar el
entero de
operación
J
K
¿es I < 2?
sí
F
no
¿es la
prioridad
P(S(I − 1),
S(I))
= 1?
sí
L
no
¿es la
prioridad
P(S(I − 1),
S(I))
= 2?
sí
M
no
¿es la
prioridad
P(S(I − 1), S(1))
= 3?
sí
F
no
¿es la
prioridad
P(S(I − 1)
S(I))
= 4?
sí
N

Código de prioridad = 1

L

P

extraer el entero A de la operación en curso

Q

hacer D = A

P

extraer el entero D de la operación anterior

Q

imprimir el símbolo de la operación R$(a)

hacer parámetro de apilamiento A = D

I

apilar el entero de la operación en curso

J

K

Código de prioridad = 2

M

P

extraer el entero de la operación en curso

Q

P

extraer el entero de la operación anterior

Q

F

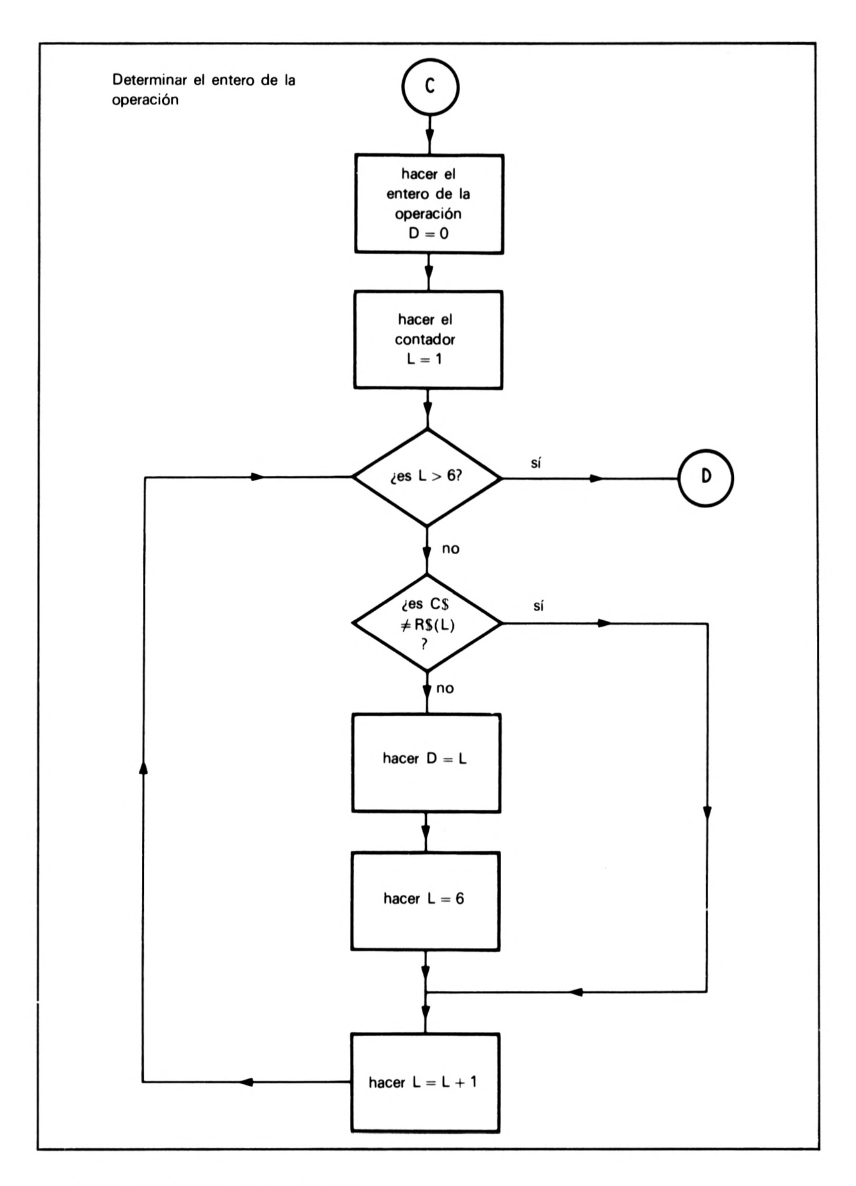
Código de prioridad = 4

N

imprimir el mensaje paréntesis mal emparejados

hacer K = LEN(E$)

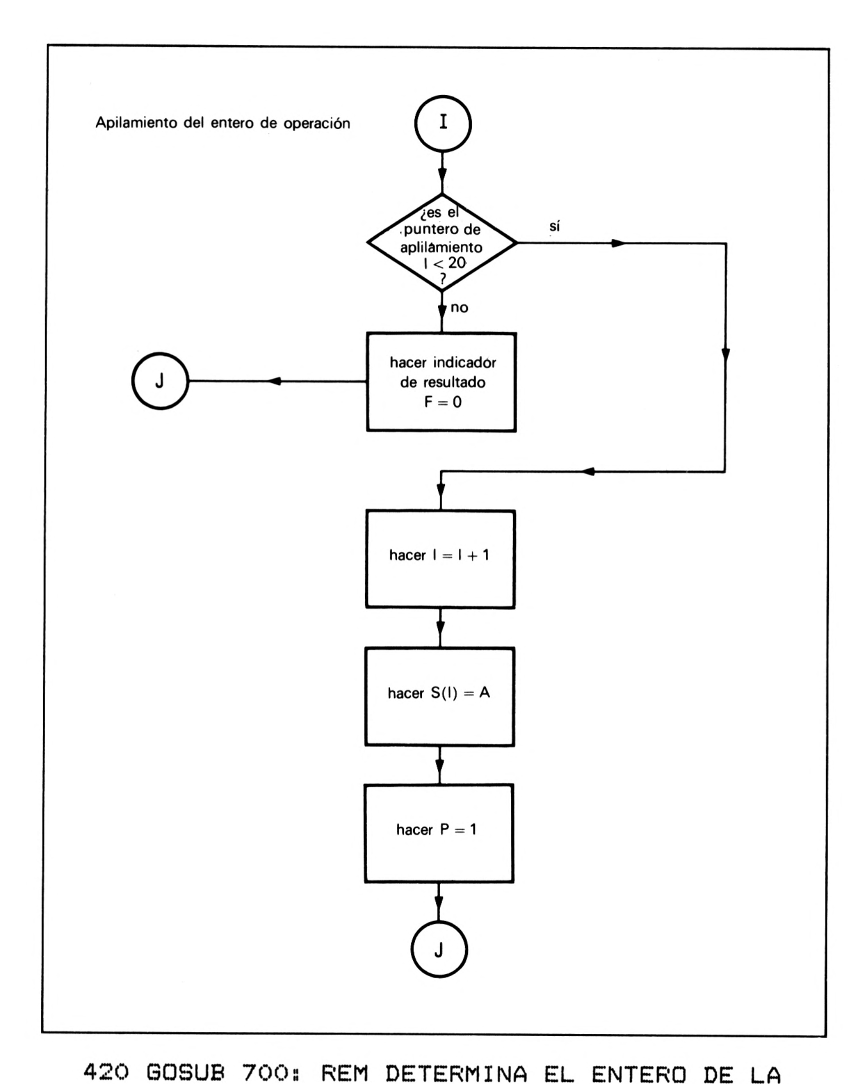
F



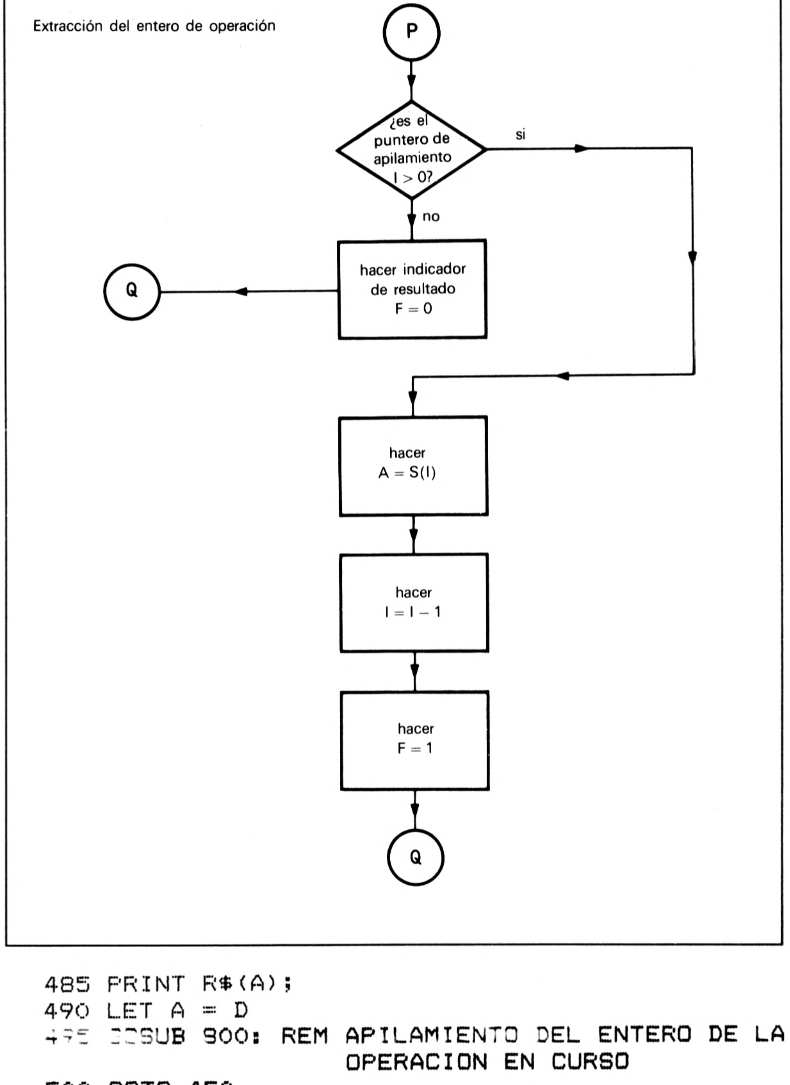

```
315 PRINT E$; "=";
320 LET E$ = "(" + E$ + ")"
325 REM
```

Análisis y conversión a notación polaca inversa

```
400 REM ANALISIS Y CONVERSION A NOTACION POLACA
        INVERSA
405 FOR K = 1 TO LEN(E$)
415 LET C$ = MID$(E$, K, 1)
```




```
420 GOSUB 700: REM DETERMINA EL ENTERO DE LA
                    OPERACION
425 IF D > 0 THEN 440
430 PRINT C$;
435 GOTO 540
440 LET A = D
445 GOSUB 800: REM APILAMIENTO DEL ENTERO DE LA
                    OPERACION
450 IF I < 2 THEN 540
455 ON P(S(I - 1), S(I)) GOTO 460, 505, 540, 525
460 REM CODIGO DE PRIORIDAD = 1
465 GOSUB 900: REM EXTRACCION DEL ENTERO DE LA
                    OPERACION EN CURSO
470 LET D = A
475 GOSUB 900: REM EXTRACCION DEL ENTERO DE LA
                    OPERACION ANTERIOR
```




```
485 PRINT R$(A);
490 LET A = D
495 GOSUB 900: REM APILAMIENTO DEL ENTERO DE LA
                   OPERACION EN CURSO
500 GOTO 450
505 REM CODIGO DE PRIORIDAD = 2
510 GOSUB 900: REM EXTRACCION DEL CODIGO DE LA
                   OPERACION EN CURSO
515 GOSUB 900: REM EXTRACCION DEL CODIGO DE LA
                   OPERACION ANTERIOR
520 GOTO 540
525 REM CODIGO DE PRIORIDAD = 4
530 PRINT "PARENTESIS MAL EMPAREJADOS"
535 LET K = LEN(E$)
540 NEXT K
545 STOP
```

Determinación del entero de la operación

```
700 REM DETERMINACION DEL ENTERO DE LA OPERACION
705 REM PARAMETROS
710 REM C$: CARACTER O SIMBOLO DE OPERACION
715 REM D: ENTERO DE LA OPERACION
720 LET D = 0
725 FOR L = 1 TO 6
730 IF C$ <> R$(L) THEN 745
735 LET D = L
740 LET L = 6
745 NEXT L
750 RETURN
```

Apilamiento del entero de la operación

```
800 REM APILAMIENTO DEL ENTERO DE LA OPERACION
805 REM PARAMETROS
810 REM A: ENTERO APILADO
815 REM F: INDICADOR DE ACIERTO/FALLO
820 IF I < 20 THEN 835
825 LET F = 0
830 RETURN
835 LET I = I + 1
840 LET S(I) = A
845 LET F = 1
850 RETURN
```

Extracción del entero de la operación

```
900 REM EXTRACCION DEL ENTERO DE LA OPERACION
905 REM PARAMETROS
910 REM A: ENTERO EXTRAIDO
915 REM F: INDICADOR DE ACIERTO/FALLO
920 IF I > 0 THEN 935
925 LET F = 0
930 RETURN
935 LET A = S(I)
940 LET I = I - 1
945 LET F = 1
950 RETURN
```

Puntos de interés

- Las verificaciones de buen o mal resultado incorporadas a los subprogramas de introducción y extracción no se usan en el programa principal. La modificación de éste para incorporarlas se deja como ejercicio.

- El programa no incluye la operación de elevación a una potencia (↑), que también queda como ejercicio.
- El subprograma encargado de determinar el entero de la operación utiliza el método de bifurcación de salida de un bucle propuesto en la sección 5.7.
- Pese a la diferente organización de la pila, los subprogramas de introducción y extracción son muy similares a los del ejemplo 18.1.

28.7 Análisis sintáctico descendente

Como ya dijimos en la sección 28.1, hay dos enfoques del análisis sintáctico: orientado hacia abajo y orientado hacia arriba o tabular. Las limitaciones impuestas por la naturaleza del Basic hacen imposible utilizar el análisis de arriba hacia abajo en la versión de referencia y en la mayoría de las versiones prácticas, que por lo general no admiten el uso de subprogramas recurrentes.

Sin embargo, ya que este segundo método es uno de los más usados en la práctica, parece razonable esbozar sus líneas generales. Además, la comparación con los métodos tabulares reforzará lo que tienen en común los dos procedimientos.

En líneas generales, el análisis sintáctico descendente consiste en la partición de una estructura en términos de subestructuras cada vez menores, que a su vez vuelven a partirse. El nivel de partición más bajo se refiere a caracteres aislados o a grupos de caracteres.

Veamos a modo de ejemplo la partición descendente de una expresión aritmética correspondiente a la tabla de prioridad de la sección 28.5; se ha utilizado una notación informal en la que la coma debe interpretarse por "seguido de":

una **expresión**	es	un **término,** una **adoperación,** una **expresión** o un **término**
un **término**	es	un **operando,** una **multioperación,** un **término** o un **operando**
un **operando**	es	1 *o* 2 *o* ... 9 *o* 0 *o* *A o B o ... o Z o* **(expresión)**
una **adoperación**	es	+ *o* −
una **multioperación**	es	* *o* /

Para expresar las especificaciones descritas con más concisión se ha desarrollado una notación especial llamada BNF.

La especificación puede parecer recurrente en el sentido de que varias estructuras se definen en términos de ellas mismas. Además, una estructura de nivel relativamente bajo —un operando— puede ser otra de nivel más alto —una expresión— encerrada entre paréntesis.

La ventaja de esta técnica es que pueden escribirse programas de análisis sintáctico en diversos lenguajes de alto nivel con un módulo por cada elemento sintáctico y una estructura general constante e igual a la descripción sintáctica. Las definiciones sintácticas recurrentes obligan a llamar a subprogramas recurrentes.

28.8 Conclusión

Hemos estudiado el análisis sintáctico y esbozado las líneas generales de las dos formas básicas de enfoçarlo. El capítulo incluye también dos programas que desarrollan en la práctica otras tantas técnicas utilizadas en uno de los enfoques generales. Los puntos más importantes son:

- El análisis sintáctico queda por lo general a cargo de programas compiladores o intérpretes, que traducen un lenguaje de alto nivel, como el Basic, a otro de bajo nivel.
- El análisis sintáctico es el proceso de reconocimiento de la estructura de un programa, o de parte de un programa, como una expresión o un número.
- Hay dos formas básicas de enfocar el análisis sintáctico: descendente y ascendente o tabular.
- El enfoque que va de lo superior a lo inferior exige definiciones recurrentes de varias estructuras y llamadas a subprogramas recurrentes capaces de identificar aquéllas. Por tanto, es incompatible con la mayor parte de las versiones del Basic y, desde luego, con la versión de referencia.

28 Ejercicio

1. Defina brevemente los siguientes términos: análisis sintáctico, enfoques descendentes y ascendentes, tabla de estados, notación polaca inversa, tabla de prioridades, operando.

2. Haga un pase de prueba del programa 28.1 con los siguientes datos:

-14
23,6
47
$3 + 42$

3. Haga un pase de prueba del programa 28.2 con los siguientes datos:

$A * (B + 2)$
$(3 - B) * (2 + A)$
$5/(K + 3 + M)$

4. Un **número decimal con signo** puede definirse de la siguiente manera:

Un número decimal con signo es un signo + o − opcional,
seguido de
una o más cifras decimales,
seguidas de
una coma decimal opcional,
seguida de
cero o más cifras decimales,
seguidas de
un carácter terminal ∇.

La tabla de estado correspondiente a esta definición es:

Estado	+	−	*Cifra*	,	*Otro carácter*	∇
1	2	2	3	error 4	error 1	error 2
2	error 3	error 3	3	error 4	error 1	error 2
3	error 3	error 3	3	4	error 1	salida 1
4	error 3	error 3	4	error 4	error 1	salida 2

a) Analice los números +43,2∇, 1,5∇, −23∇, −0,4∇ y −4, 2, 6∇ por medio de la tabla.
b) Escriba mensajes adecuados para las situaciones de error y las salidas.
c) Modifique o vuelva a escribir el programa 28.1 para analizar números decimales con signo tal como aquí se han definido.

5. Un número se escribe en términos de una potencia de diez de la forma siguiente:

$1{,}3 \times 10^5$
$0{,}6667 \times 10^{-4}$

En la mayor parte de los lenguajes de programación, la potencia de diez se representa mediante la letra **E**:

1,3 E5
0,6667 E − 4

a) Escriba una definición de la forma normal de un número de este tipo.
b) Establezca la tabla de estados equivalente a la definición.
c) Modifique o vuelva a escribir el programa ejemplo 28.1 para analizar la sintaxis de la forma normal de tales números.

6. Partiendo de la versión original o de una modificación del programa ejemplo 28.1:
a) Incluya el conjunto entrada-análisis-salida en un bucle, de manera que el usuario pueda analizar una sucesión de números.
b) Presente en pantalla unas instrucciones para el usuario adecuadamente organizadas.
c) Escriba la documentación para el programador y la guía para el usuario correspondiente al programa modificado.

7. La operación de elevación a una potencia, simbolizada ↑ en muchos lenguajes de programación, tiene prioridad sobre las otras cuatro.
a) Amplíe la tabla de prioridades de la sección 28.5 para incluir la operación ↑.
b) Modifique el programa ejemplo 28.2 para incluir en él la operación ↑.

8. El programa ejemplo 28.2 no somete las expresiones aritméticas a una verificación muy estricta de su validez. Modifíquelo para que lleve a cabo las siguientes comprobaciones:
a) Operando incorrecto (ni entero decimal ni letra mayúscula).
b) Dos símbolos de operación seguidos en una misma expresión.
c) Rebosamiento positivo o negativo de la pila.

9. Partiendo de la forma original del programa ejemplo 28.2 o de una modificación del mismo:
a) Incluya el conjunto entrada-análisis-salida en un bucle de manera que el usuario pueda analizar una serie de expresiones.
b) Presente en pantalla unas instrucciones para el usuario adecuadamente organizadas.
c) Redacte la documentación para el programador y la guía para el usuario de la versión modificada.

10. La sintaxis de una línea de programa escrito en Basic puede especificarse de manera informal como sigue:

una **línea de programa** es un número de línea formado por una o más cifras decimales,
seguido de
una palabra de instrucción formada por hasta seis letras mayúsculas,
seguida opcionalmente de
una serie de caracteres alfanuméricos.

Observe que no se trata de una definición precisa, porque no tiene en cuenta la sintaxis de la línea tras la palabra de instrucción.
a) Construya la tabla de estados correspondiente a la definición.
b) Escriba un programa capaz de analizar la estructura de una o más líneas de un programa en Basic por medio de la tabla de estados.


RESET
RETURN

29 Simulación de un ordenador

Un ordenador cualquiera tiene la propiedad de poder comportarse como otro diferente. Esto puede conseguirse por **emulación,** que consiste en la modificación del soporte lógico de nivel muy bajo o, más fácilmente, por **simulación,** que consiste en ejecutar un programa que hace al ordenador en cuestión comportarse como si fuese otro. La simulación tiene varias aplicaciones, como la comprobación del diseño de una nueva unidad de proceso o la exposición con fines didácticos de la estructura de la misma y del lenguaje máquina.

Este capítulo se centra en la segunda de las aplicaciones mencionadas. La forma más didáctica de proceder es crear un **modelo de ordenador** que contenga los elementos esenciales de una máquina procesadora, pero sin las complejidades y limitaciones propias de la misma. El modelo se diseña teniendo en cuenta el nivel del curso de informática en que pretende utilizarse.

Escribir un programa que simule un modelo de ordenador con sencillez y eficacia no es tarea fácil. En este capítulo definiremos un **modelo de ordenador sencillo (MOS),** adecuado para cursos elementales de introducción a la informática, y a continuación diseñaremos un programa que lo ejecute en la práctica.

En el ejercicio propuesto al final del capítulo se harán algunas sugerencias para mejorar el modelo de partida y para crear otros más complejos. Se persigue aquí un doble objetivo: escribir un programa que simule un modelo dado de ordenador y diseñar ese modelo.

Aunque se ha procurado que el texto sea completo en sí mismo, se comprenderá más cabalmente si se tienen algunos conocimientos de estructura de unidades de proceso y de lenguajes de bajo nivel.

29.1

Un modelo de ordenador sencillo: el MOS

Este **modelo de ordenador sencillo (MOS)** se ha diseñado para enseñar los conceptos más elementales de arquitectura de máquinas procesadoras y lenguajes de bajo nivel. Más concretamente, se describirán los conceptos y características siguientes:

- Una unidad central de proceso consta de **unidad de control, unidad aritmética y lógica** y **memoria central.**
- Cada una de esas unidades está formada por varios **registros** que almacenan elementos de información específicos y por una serie de circuitos **procesadores** que llevan a cabo operaciones concretas.
- Los registros están conectados por un **canal** *(bus)* encargado del transporte de la información.
- La unidad central de proceso se controla directamente por medio de programas escritos en **lenguaje máquina.**
- Un lenguaje máquina consta de una serie de **instrucciones** simples ejecutada cada una de ellas por uno o más registros y por los circuitos procesadores asociados a ellos.
- La secuencia de operaciones necesaria para ejecutar una instrucción de máquina se llama **ciclo de instrucción** del procesador.
- A la memoria central se accede por medio de una **dirección** que **localiza** un dato o una instrucción de máquina.

La disposición de los registros del MOS y su lenguaje máquina materializan estas características de la forma más sencilla posible.

29.2

Disposición de los registros del MOS

La figura 29.1 ilustra la organización de los registros del MOS y de algunos de sus circuitos de proceso. Los primeros contienen de 4 a 8 bits y están conectados por un **canal** *(bus)* de 8 bits de capacidad.

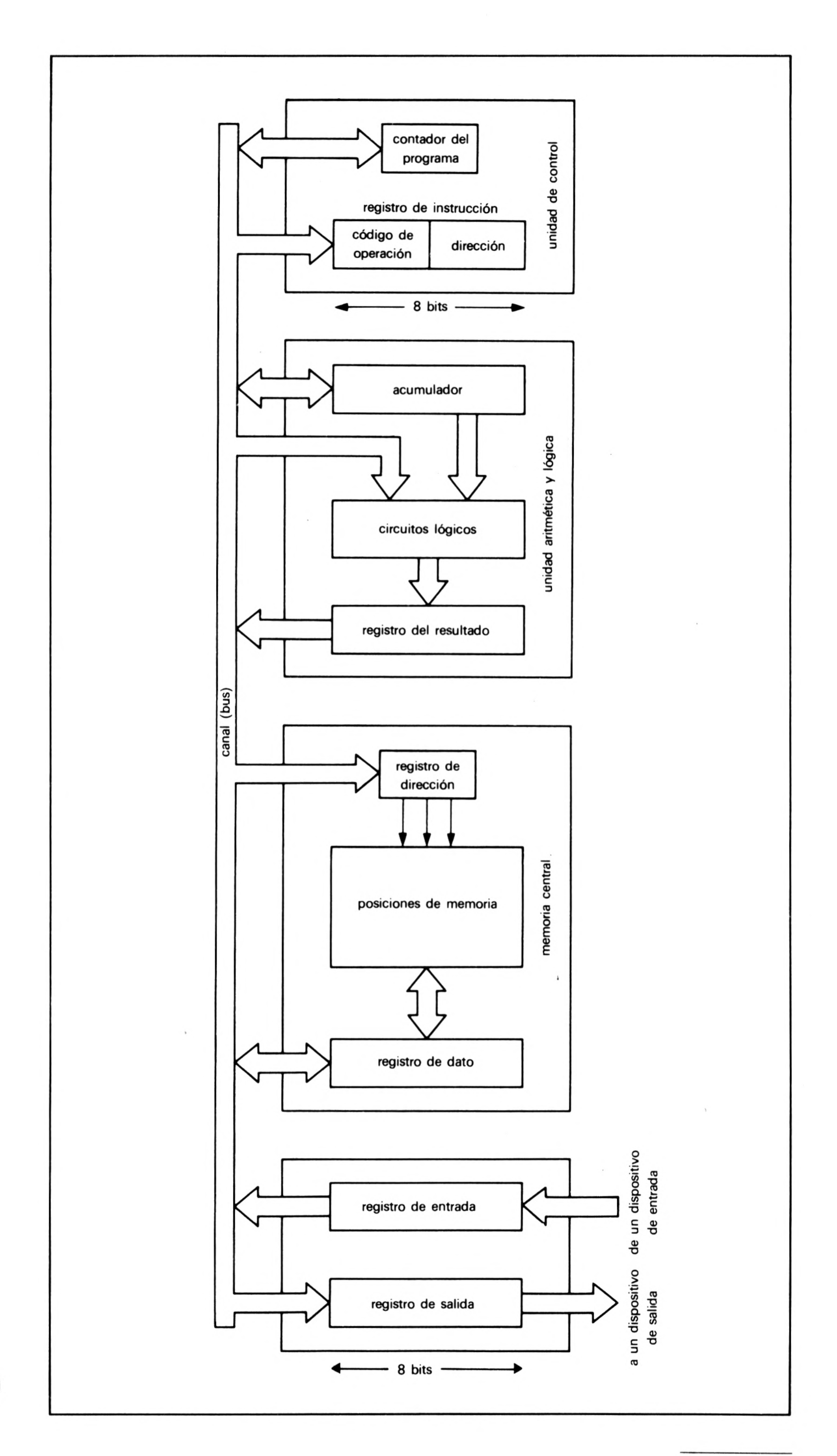


Figura 29.1
Disposición de los registros del MOS

La **memoria central** tiene 16 **posiciones** identificadas por las direcciones 0 a 15. En cada una de las posiciones cabe un dato o una instrucción de máquina de 8 bits. La dirección de la posición de memoria a la que se accede en un momento dado se encuentra en un **registro de dirección** de 4 bits; hay además un **registro de dato** de 8 bits que alberga el elemento transferido a o extraído de una posición de memoria.

La **unidad aritmética y lógica** tiene un **acumulador** de 8 bits que alberga el dato sometido a tratamiento. Los circuitos lógicos reciben entradas procedentes del acumulador y del bus y proporcionan un **registro de resultado** de 8 bits. Los circuitos lógicos mencionados ejecutan operaciones sencillas, como la adición o la negación.

La **unidad de control** tiene un **contador de programa** de 4 bits que almacena la dirección de la instrucción de máquina que va a ejecutarse y un **registro de instrucción** de 8 bits que alberga una instrucción de máquina durante su ejecución. Cada una de estas instrucciones consta de un **código de operación** de 4 bits, que determina el tipo de operación, y, en muchos casos, una **dirección** de 4 bits encargada de localizar el dato que ha de utilizarse.

El **registro de entrada** de 8 bits recibe los datos procedentes de un dispositivo de entrada, y el de **salida,** también de 8 bits, se encarga de transferir los resultados a otro dispositivo, esta vez de salida.

Como puede comprobarse, el MOS, pese a su sencillez, contiene los elementos esenciales de una unidad central de proceso: control, circuitos de tratamiento, memoria y dispositivos de entrada y salida.

29.3

Lenguaje máquina del MOS

Este lenguaje refleja la disposición de los registros. Cada instrucción ejecuta una operación simple y única que, por lo general, afecta al acumulador y, en muchos casos, a un dato de la memoria.

Cada instrucción contiene un código de operación de 4 bits, lo que limita el número máximo de operaciones a 16. Por la misma razón, como la dirección de la instrucción tiene también 4 bits, da acceso a cualquiera de las 16 posiciones de memoria.

Las instrucciones y las direcciones se identifican por medio de una **cifra hexadecimal** (que representa 4 bits). La figura 29.2 recoge el **juego** completo **de instrucciones.** Observe que no todas tienen dirección y que de los 16 códigos de operaciones disponibles sólo se han utilizado 13.

código de operación (hexadecimal)	palabra código	interpretación
0	**PET**	detener.
1	**INT**	reproducir el registro de entrada en el acumulador.
2	**SAL**	reproducir el acumulador en el registro de salida.
3X	**LOA X**	reproducir la posición de memoria **X** en el acumulador.
4X	**STD X**	reproducir el acumulador en la posición de memoria **X**.
5X	**SUM X**	sumar la posición **X** al acumulador.
6	**CLA**	borrar el acumulador.
7	**INC**	sumar 1 al acumulador.
8	**NEG**	cambiar el signo del acumulador.
9X	**BRN X**	bifurcar a la instrucción de la posición **X**.
AX	**BZE X**	bifurcar si el acumulador vale cero.
BX	**BGT X**	bifurcar si el acumulador es positivo.
CX	**BNG X**	bifurcar si el acumulador es negativo.

Observaciones:
La letra **X** representa una cifra hexadecimal.
Los códigos de operación D a F no se utilizan en esta versión del MOS.

Figura 29.2
Juego de instrucciones del MOS

29.4 Ejemplo de programa en el lenguaje máquina del MOS

Este sencillo programa tiene por objeto ilustrar el funcionamiento del lenguaje máquina del MOS. El programa en cuestión acepta la entrada de dos números de 8 bits, resta uno a otro y lleva a la salida la diferencia.

Si se trabaja en aritmética **complementaria de base dos,** la resta se efectúa calculando el complemento del segundo número (cambiándole el signo y sumándole 1) y sumando el resultado al primer número. El programa está cargado en una parte de la memoria central del MOS, y empieza en la dirección 0.

Dirección (hexadecimal)	*Instrucción* (hexadecimal)	*Código*	*Interpretación*
0	**1**	**INT**	Introducir el primer número.
1	**4 8**	**ALM 8**	Almacenarlo en la posición 8.
2	**1**	**INT**	Introducir el segundo número.
3	**8**	**NEG**	Cambiar el signo del acumulador.

4	**7**	**INC**	Sumar 1 al acumulador (produce el complemento del segundo número).
5	**5 8**	**SUM 8**	Sumar el primer número (produce la diferencia entre los dos).
6	**2**	**SAL**	Llevar el resultado a la salid...
7	**0**	**DET**	Detenerse.
8			Posición de memoria para el primer número.

Observe que cuando una instrucción no lleva dirección, se supone que la parte correspondiente a la misma es cero, aunque se omite por razones de claridad.

29.5 Ciclo de instrucción del MOS

Se llama ciclo de instrucción a la secuencia de operaciones necesarias para ejecutar completamente una instrucción de máquina. Por lo general tiene dos fases: **cargar** y **ejecutar.** En el caso del MOS, estas fases se desarrollan como sigue:

- Utilizar la dirección del contador del programa para **cargar** en el registro de instrucción la siguiente instrucción de máquina.
- Sumar 1 al contador del programa y dejarlo así listo para la instrucción siguiente.
- **Ejecutar** la instrucción en el registro de la misma. Utilizar el código de operación para determinar su naturaleza y la dirección para localizar el dato en la memoria central.

Observe que el ciclo de instrucción supone traer una instrucción y, en algunos casos, traer o almacenar un dato. Cada una de estas actividades de acceso a la memoria central supone a su vez una secuencia adicional de operaciones conocida como **ciclo de memoria.**

La figura 29.3 ilustra a grandes rasgos el ciclo de instrucción del MOS. Más adelante, con ocasión del diseño del programa de simulación, se hará una versión más detallada.

29.6 Programa ejemplo 29.1

Dado que el MOS es un modelo diseñado con fines didácticos, el programa de simulación no puede limitarse a la carga y ejecución de

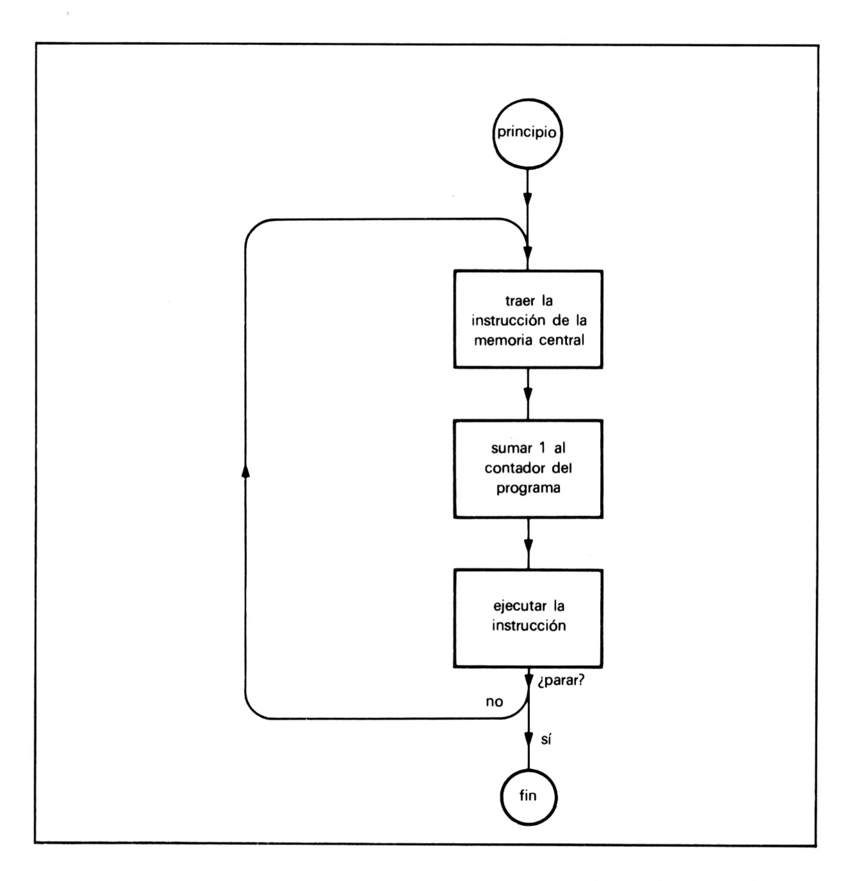


Figura 29.3
Ciclo de instrucción del MOS

otros programas, sino que además debe satisfacer los siguientes objetivos:

- Cargar un programa en la memoria central del MOS y verificarlo durante esta operación.
- Visualizar el contenido en curso de los registros y la memoria central del MOS.
- Ejecutar un programa, presentando en pantalla los contenidos de los registros tras cada instrucción (la ejecución puede ser continua o instrucción por instrucción).
- Permitir la interrupción y ulterior puesta en marcha de un programa que avanza paso a paso tras cualquier instrucción.
- Permitir mejoras como la inclusión de un programa ensamblador (véase sección 29.7).

Diseño del programa

Como muchos otros de los estudiados en el libro, este programa está organizado en varios niveles, de forma que cualquiera de ellos

está controlado por los que tiene por encima de sí. En el más alto se encuentra el **módulo de mando,** que responde a las órdenes **CARGAR, VISUALIZAR, EJECUTAR** y **PARAR** y tiene prevista la adición más adelante de algunas otras. Cada orden determina la transferencia del control a un **módulo de operación** que, junto con el de **inicialización,** constituyen el siguiente nivel del programa.

El módulo de inicialización organiza las estructuras de datos necesarias y ocupa con ceros todos los registros y posiciones de memoria. También se encarga de presentar una serie de instrucciones dirigidas al usuario.

El módulo **CARGAR** supervisa la introducción en el MOS del programa escrito en lenguaje máquina, verifica cada una de las instrucciones y lo carga en la memoria central.

El módulo **VISUALIZAR** imprime o presenta en pantalla los contenidos de todos los registros y posiciones de memoria. La escritura del mismo queda como ejercicio para el lector.

El módulo **EJECUTAR** simula una secuencia de ciclos de máquina del MOS. Cada instrucción cuenta con un subprograma al que se transfiere el control en caso de que sea identificada. Las operaciones como leer o escribir en la memoria principal y convertir la base de los sistemas de numeración se encargan a módulos de un nivel inferior.

El módulo **PARAR** detiene el programa.

El nivel inferior de la estructura general del programa alberga una serie de módulos de servicio encargados de operaciones como el cambio de base de los sistemas de numeración y el ciclo de memoria. La estructura completa del programa de simulación se ilustra en la figura 29.4.

Representación de los datos

Todos los datos e instrucciones de máquina se almacenan en el programa como cifras hexadecimales, lo que supone representar los registros mediante variables alfanuméricas y las posiciones de memoria mediante matrices de la misma naturaleza. Sin embargo, la búsqueda de direcciones, la suma y otras operaciones deben realizarse en sistema decimal. Por tanto, hay que crear módulos de servicio para hacer las necesarias transformaciones entre las bases 16 y 10 (se usa para ello una tabla de cifras hexadecimales).

Estructura detallada del programa

Decididas ya la estructura general y la representación de los datos pasaremos a la siguiente fase del diseño, que consistirá en detallar algo más parte de los módulos, concretamente los llamados **CARGAR** y **EJECUTAR**.

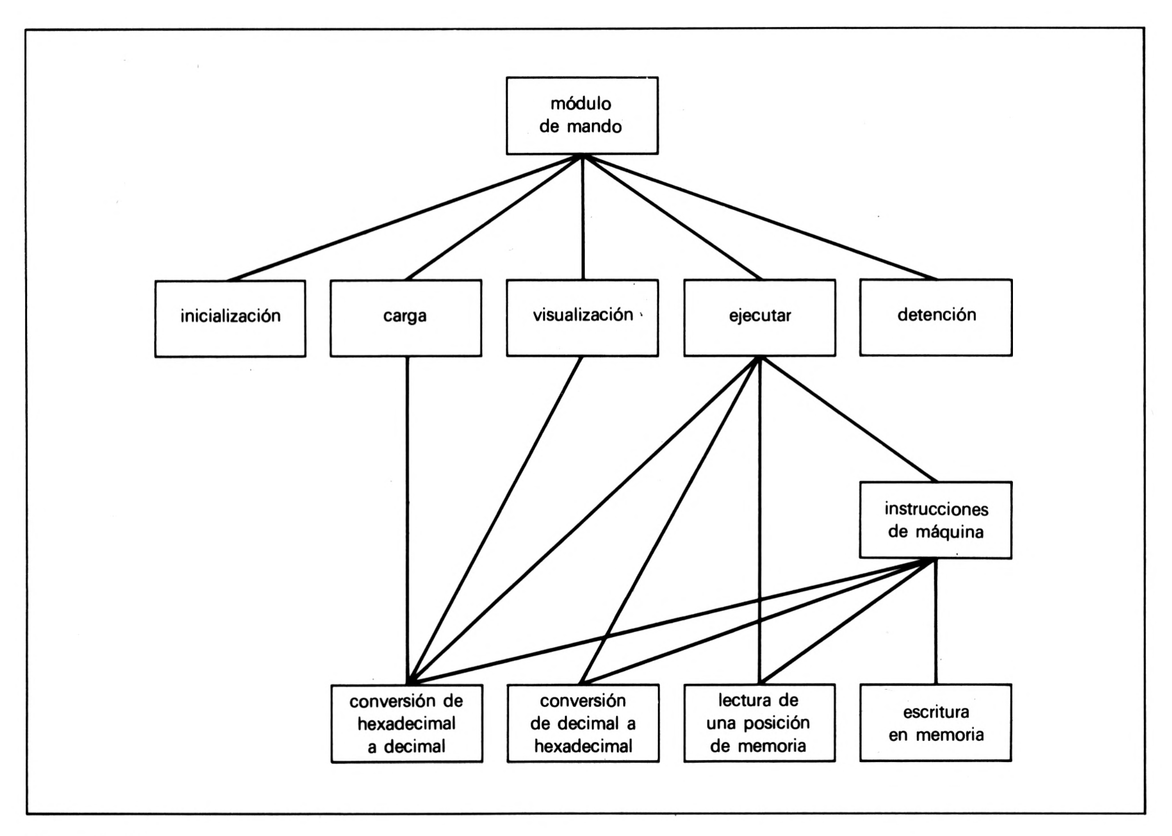


Figura 29.4
Estructura general del programa de simulación del MOS

CARGAR

Por razones de simplicidad, todos los programas se cargan partiendo de la posición cero de la memoria. El algoritmo que rige la operación es:

Repetir el proceso:

Repetir el proceso:

Introducir una instrucción del programa (dos cifras hexadecimales).

Convalidar la instrucción transformando las cifras al sistema decimal.

Hasta que la instrucción sea válida.

Cargar la instrucción en una posición de la memoria.

Hasta localizar el indicador de final de entrada o hasta que la memoria se llene.

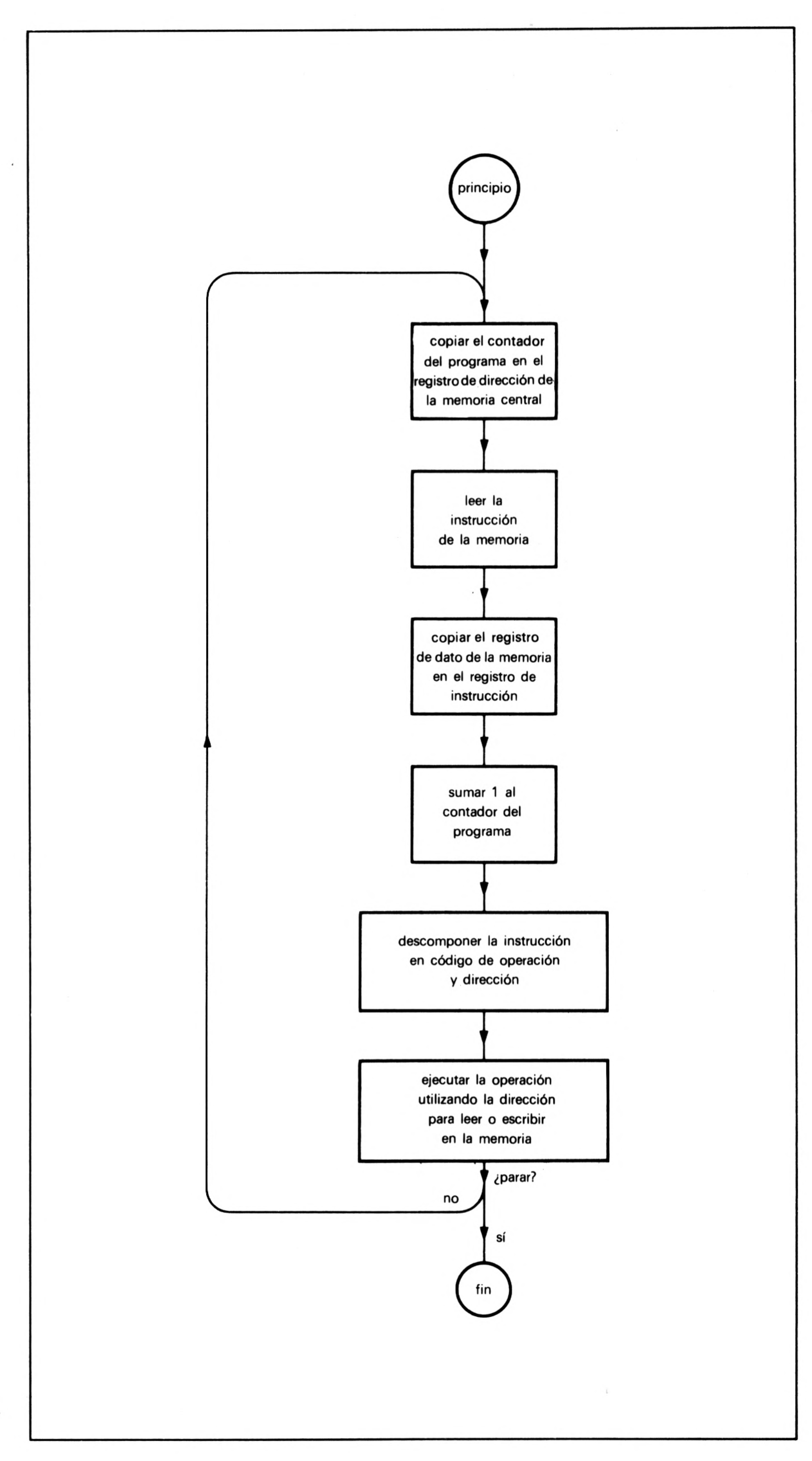


Figura 29.5
Ciclo de instrucción MOS detallado

EJECUTAR

El módulo **EJECUTAR** simula con cierto detalle el ciclo de instrucción del MOS. Por tanto, en la figura 29.5 se ha vuelto a ilustrar ese ciclo en términos de las transferencias de datos entre registros. El algoritmo que rige el módulo es el siguiente:

> Determinar la dirección de partida y el modo de avance (continuo o paso a paso).
>
> Visualizar encabezamientos y contenidos de registros.
>
> Repetir el proceso.
>
>> Reproducir el contenido del contador del programa en el registro de direccionamiento de la memoria.
>>
>> Leer la instrucción de la memoria.
>>
>> Reproducir el registro de datos de la memoria en el registro de instrucción.
>>
>> Sumar 1 al contador del programa.
>>
>> Descomponer la instrucción en código de operación y dirección.
>>
>> Transferir el control al subprograma correspondiente al código de operación para que ejecute ésta.
>>
>> Visualizar el contenido actual de los registros.
>>
>> Si la modalidad es paso a paso, introducir un carácter para continuar.
>
> Hasta alcanzar el final del programa o hasta que se introduzca una instrucción de interrupción.

El programa de simulación está ya especificado con detalle suficiente y puede escribirse. Además del módulo **VISUALIZAR**, se han dejado como ejercicios varios subprogramas de instrucciones de máquina y las instrucciones iniciales para el usuario.

Variables

C$	Contador del programa.
I$	Registro de instrucción: **O$** código de operación.
	D$ dirección.
A$	Acumulador: **A** equivalente decimal.
R$	Registro del resultado.
E$	Registro de dirección de la memoria.
S$(15)	Posiciones de la memoria.

G$ Registro de datos de la memoria; **G**: equivalente decimal.
N$ Registro de entrada.
U$ Registro de salida.
H$ Cifra hexadecimal (parámetro de subprograma).
D Cifra decimal (parámetro de subprograma).
X$(15) Tabla de cifras hexadecimales.
I, J, K Contadores de bucle.
M$ Orden.
T$ Instrucción de máquina (entrada) y dirección de partida (entrada).
B$ Modalidad de paso (continuo o paso a paso).
Q$ Carácter introducido para continuar en la modalidad paso a paso.

Diagrama de flujo

Como la mayor parte de los módulos siguen una pauta ya conocida, sólo se ha trazado el diagrama de flujo para el llamado **EJECUTAR**, que se ilustra en la figura 29.6.

Programa

```
1000 REM PROGRAMA EJEMPLO 29.1
1005 REM PROGRAMA DE SIMULACION MOS
1010 REM
```

Inicialización

```
1100 REM INICIALIZACION
1105 DIM X$(15), S$(15)
1110 FOR K = 0 TO 15
1115 LET S$(K) = "00"
1120 READ X$(K)
1125 NEXT K
1130 DATA "0", "1", "2", "3", "4", "5", "6", "7"
1135 DATA "8", "9", "A", "B", "C", "D", "E", "F"
1140 C$ = "0": I$ = "00": A$ = "00": N$ = "00":
     U$ ="00"
1145 PRINT "PROGRAMA DE SIMULACION MOS"
1150 PRINT
1155 REM LAS INSTRUCCIONES DEL USUARIO
1160 REM SE DEBEN ESCRIBIR COMO EJERCICIO
1165 REM
```

Módulo de orden

```
1200 REM MODULO DE ORDENES
1205 PRINT "ORDEN?";
```

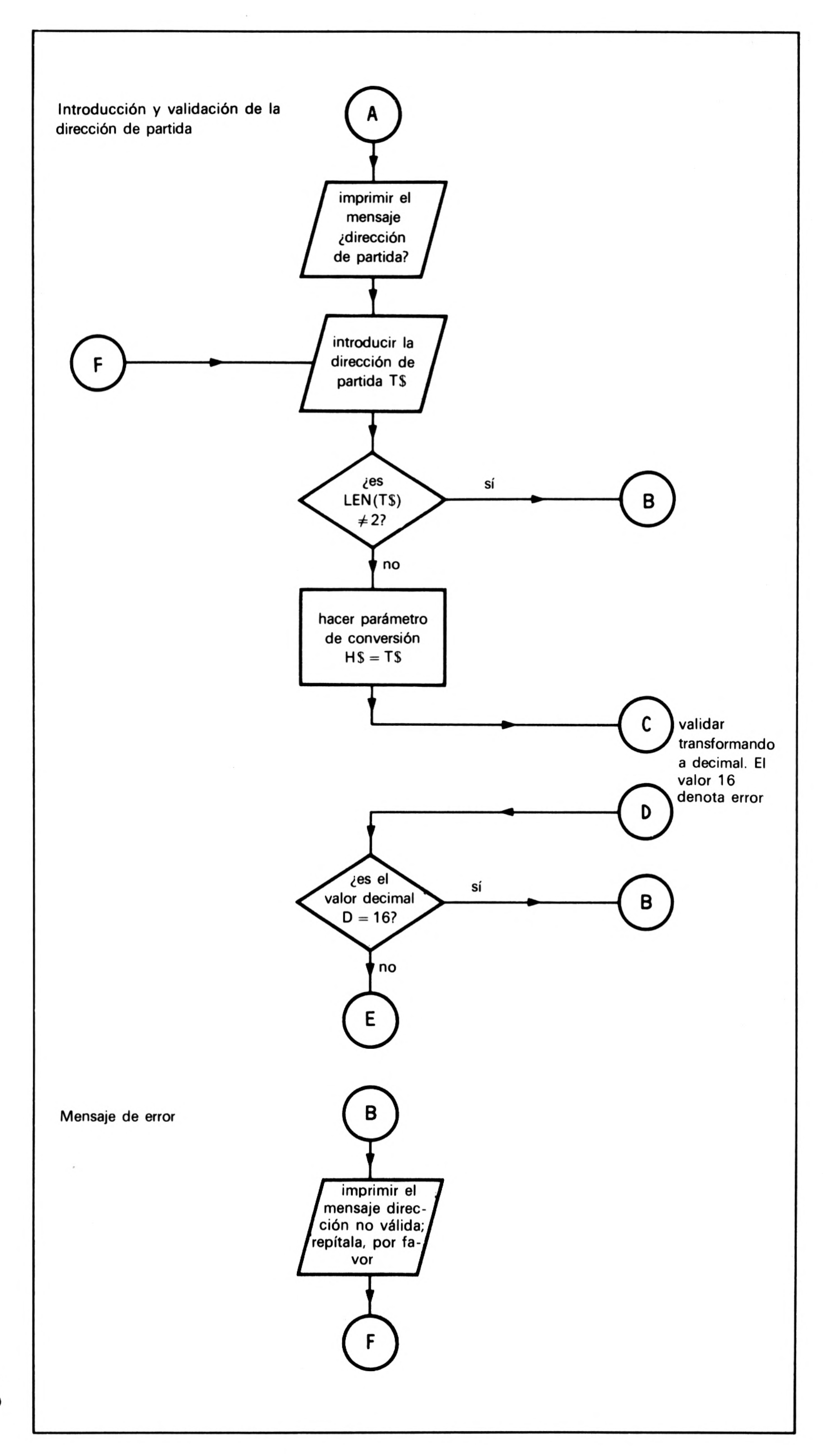
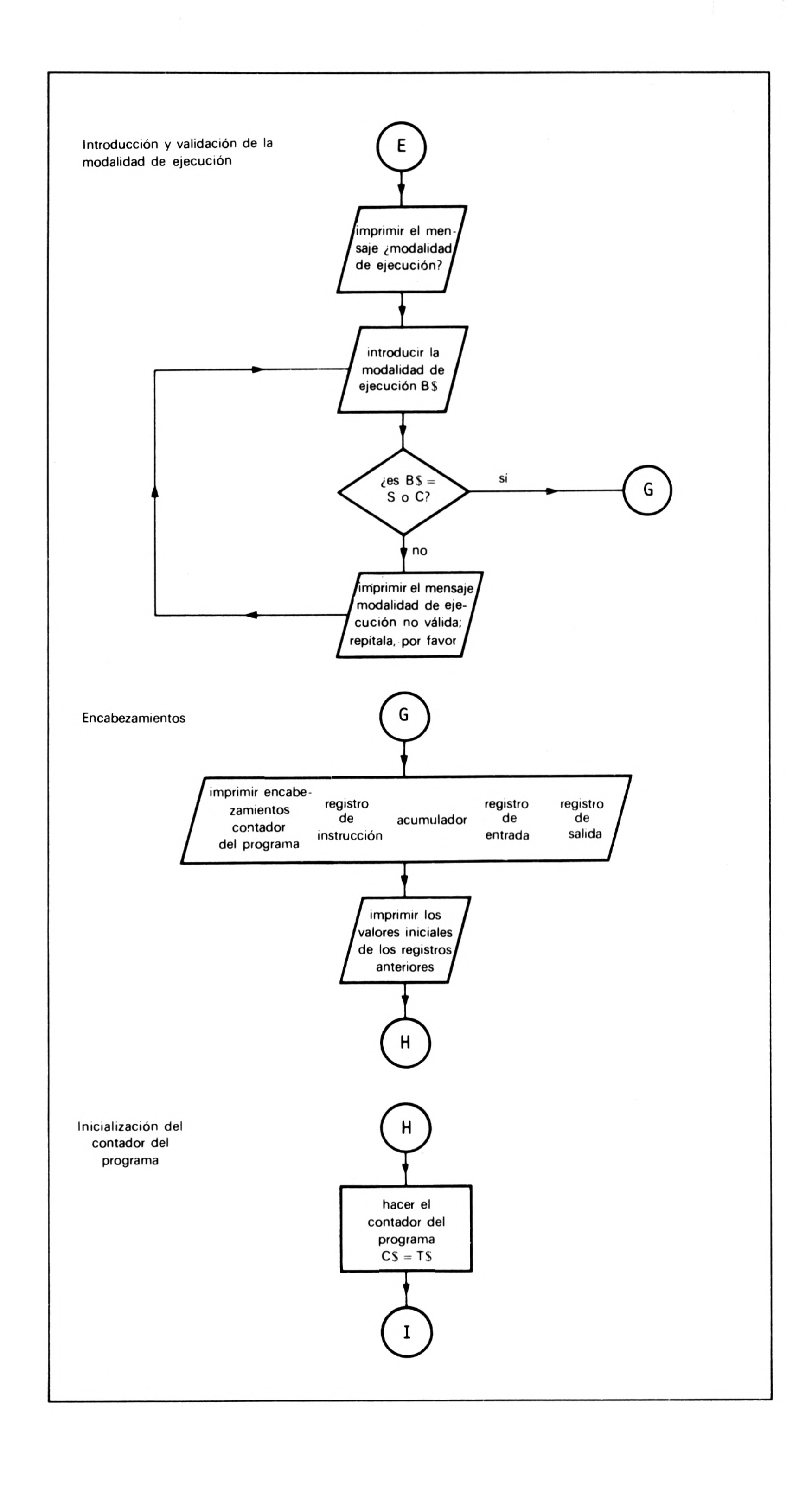
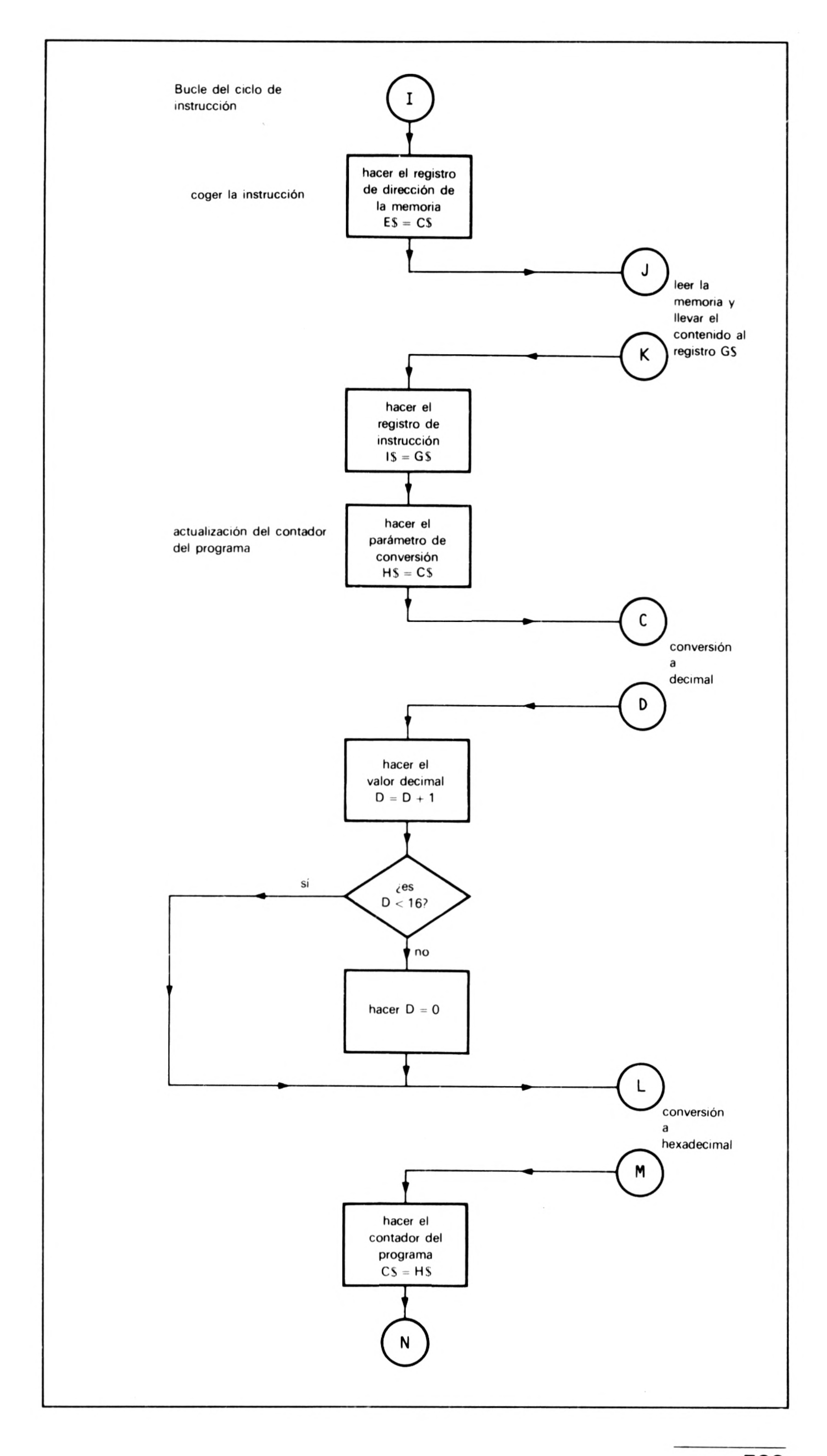


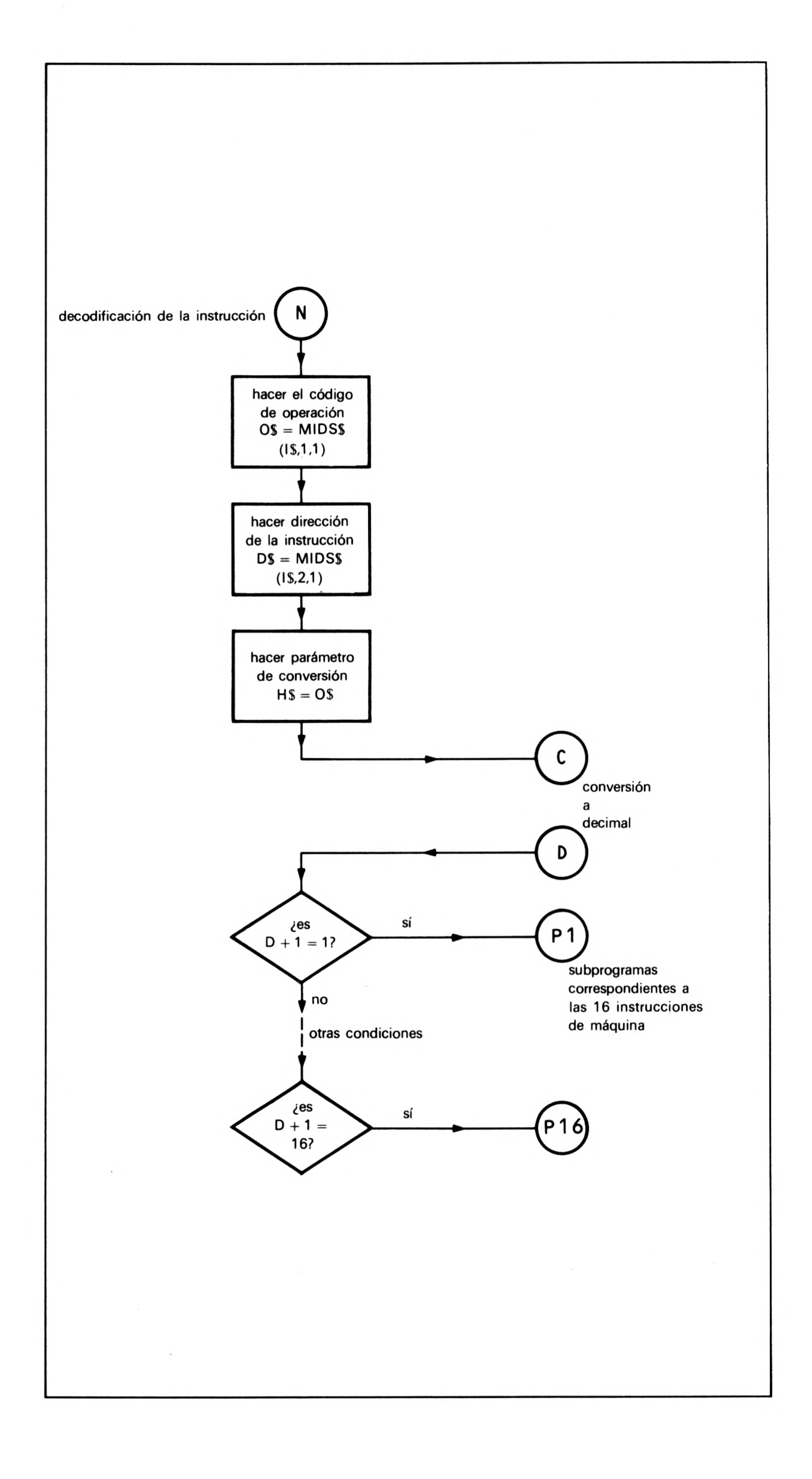
Figura 29.6
Flujo de control del módulo EJECUTAR del programa ejemplo 29.1

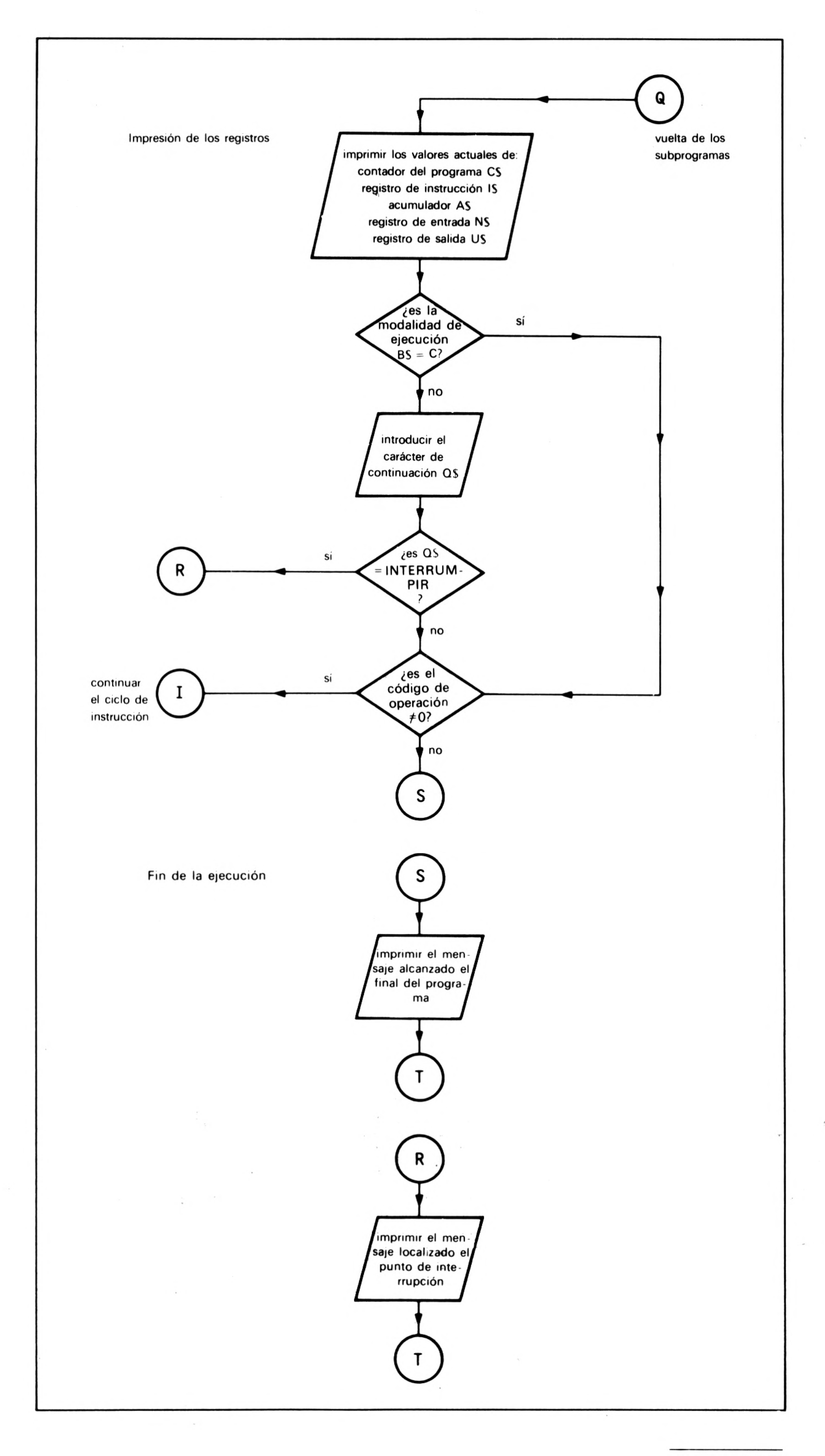


Introducción y validación de la modalidad de ejecución
E
imprimir el mensaje ¿modalidad de ejecución?
introducir la modalidad de ejecución B$
¿es B$ = S o C?
sí
G
no
imprimir el mensaje modalidad de ejecución no válida; repítala, por favor
Encabezamientos
G
imprimir encabezamientos contador del programa
registro de instrucción
acumulador
registro de entrada
registro de salida
imprimir los valores iniciales de los registros anteriores
H
Inicialización del contador del programa
H
hacer el contador del programa C$ = T$
I



Bucle del ciclo de instrucción
I
coger la instrucción
hacer el registro de dirección de la memoria E$ = C$
J
leer la memoria y llevar el contenido al registro G$
K
hacer el registro de instrucción I$ = G$
actualización del contador del programa
hacer el parámetro de conversión H$ = C$
C
conversión a decimal
D
hacer el valor decimal D = D + 1
sí
¿es D < 16?
no
hacer D = 0
L
conversión a hexadecimal
M
hacer el contador del programa C$ = H$
N


decodificación de la instrucción
N
hacer el código de operación O$ = MID$$ (I$,1,1)
hacer dirección de la instrucción D$ = MID$$ (I$,2,1)
hacer parámetro de conversión H$ = O$
C
conversión a decimal
D
¿es D + 1 = 1?
sí
P1
subprogramas correspondientes a las 16 instrucciones de máquina
no
otras condiciones
¿es D + 1 = 16?
sí
P16



Q
vuelta de los subprogramas
Impresión de los registros
imprimir los valores actuales de:
contador del programa C$
registro de instrucción I$
acumulador A$
registro de entrada N$
registro de salida U$
¿es la modalidad de ejecución B$ = C?
sí
no
introducir el carácter de continuación Q$
¿es Q$ = INTERRUM-PIR ?
sí
R
no
¿es el código de operación ≠0?
sí
I
continuar el ciclo de instrucción
no
S
Fin de la ejecución
S
imprimir el mensaje alcanzado el final del programa
T
R
imprimir el mensaje localizado el punto de interrupción
T

```
1210 INPUT M$
1215 IF M$ = "CARGAR" THEN 1300
1220 IF M$ = "VISUALIZAR" THEN 1500
1225 IF M$ = "EJECUTAR" THEN 1700
1230 REM ESPACIO PARA COMANDOS ADICIONALES
1250 IF M$ = "PARAR" THEN 9900
1255 PRINT "ORDEN NO RECONOCIDA. POR FAVOR REPITA
          LA"
1260 GOTO 1210
1265 REM
```

Módulo CARGAR

```
1300 REM MODULO DE CARGA
1305 PRINT "TECLEE LAS INSTRUCCIONES DEL
          LENGUAJE MAQUINA"
1310 PRINT "DE UNA EN UNA"
1315 PRINT "FINALICE SU INTRODUCCION CON LOS
          CARACTERES **"
1320 PRINT
1325 LET K1 = 0
1330 INPUT T$
1335 IF T$ = "**" THEN 1420
1340 REM VALIDACION
1345 IF LEN(T$) <> 2 THEN 1385
1350 LET H$ = MID$(T$, 1, 1)
1355 GOSUB 3900: REM CONVERSION A DECIMAL
1360 IF D = 16 THEN 1385
1365 LET H$ = MID$(T$, 2, 1)
1370 GOSUB 3900: REM CONVERSION A DECIMAL
1375 IF D = 16 THEN 1385
1380 GOTO 1395
1385 PRINT "INSTRUCCION INCORRECTA. POR FAVOR
          REPITALA"
1390 GOTO 1330
1395 REM ALMACENAMIENTO DE INSTRUCCION CORRECTA
1400 LET S$(K1) = T$
1405 LET K1 = K1 + 1
1410 IF K1 < 16 THEN 1330
1415 PRINT "MEMORIA COMPLETA"
1420 PRINT "FIN DE LA INTRODUCCION"
1425 PRINT
1430 GOTO 1200
1435 REM
```

Módulo VISUALIZAR

```
1500 REM EL MODULO DE VISUALIZACION
1505 REM SE DEBE ESCRIBIR COMO EJERCICIO
1690 GOTO 1200
1695 REM
```

Módulo EJECUTAR

```
1700 REM MODULO DE EJECUCION
1705 PRINT "DIRECCION DE PARTIDA?";
1710 INPUT T$
1715 REM VALIDACION
1720 IF LEN(T$) <> 1 THEN 1745
1725 LET H$ = T$
1730 GOSUB 3900: REM CONVERSION A DECIMAL
1735 IF D = 16 THEN 1745
1740 GOTO 1755
1745 PRINT "DIRECCION INCORRECTA. POR FAVOR REPI
           TALA"
1750 GOTO 1710
1755 PRINT "MODO DE EJECUCION- C PARA CONTINUAR"
1760 PRINT "S PARA PASO A PASO?";
1765 INPUT B$
1770 IF B$ = "S" OR B$ = "C" THEN 1785
1775 PRINT "MODO INCORRECTO. POR FAVOR REPITALO";
1780 GOTO 1765
1785 LET C$ = T$
1790 REM
1800 REM VISUALIZACION  DE CABECERAS
1805 PRINT
1810 PRINT "PROGRAMA"; TAB(13); "INSTR"; TAB(21);
           "ACUMU";
1815 PRINT TAB(29); "ENTRADA"; TAB(37); "SALIDA"
1820 PRINT "CONTADOR"; TAB(13); "REG"; TAB(21);
           "LADOR"
1825 PRINT TAB(29); "REG"; TAB(37); "REG"
1830 PRINT
1835 PRINT C$; TAB(13); I$; TAB(21); A$; TAB(29);
           N$; TAB(37); U$
1840 REM
1900 REM BUCLE DEL CICLO DE INSTRUCCIONES
1902 REM COGER LA INSTRUCCION
1905 LET E$ = C$
1910 GOSUB 3700: REM LECTURA DE LA MEMORIA
1915 LET I$ = G$
1917 REM ACTUALIZACION DEL CONTADOR DEL PROGRAMA
1920 LET H$ = C$
1925 GOSUB 3900: REM CONVERSION A DECIMAL
1930 LET D = D + 1
1935 IF D < 16 THEN 1945
1940 LET D = 0
1945 GOSUB 4000: REM CONVERSION A HEXADECIMAL
1950 LET C$ = H$
1952 REM DECODIFICACION DE LA INSTRUCCION
1955 LET O$ = MID$(I$, 1, 1)
1960 LET D$ = MID$(I$, 2, 1)
1965 LET H$ = O$
1970 GOSUB 3900: REM CONVERSION A DECIMAL
1975 REM SUBPROGRAMAS QUE EJECUTAN LAS
         INSTRUCCIONES EN LENGUAJE MAQUINA
```

```
1980 REM DE ACUERDO CON EL CODIGO DE OPERACION
1985 ON D + 1 GOSUB 2200, 2300, 2400, 2500, 2600,
                    2750, 2800, 2900, 3000, 3100,
                    3200, 3300, 3400, 3500, 3600
1990 PRINT C$; TAB(8); I$; TAB(16); A$; TAB(24);
          N$; TAB(32); U$
1995 IF B$ = "C" THEN 2010
2000 INPUT Q$
2005 IF Q$ ="INTERRUMPIR" THEN 2025
2010 IF O$ <> "0" THEN 1900
2015 PRINT "SE HA LLEGADO AL FINAL DEL PROGRAMA"
2020 GOTO 1200
2025 PRINT "LOCALIZADO EL PUNTO DE INTERRUPCION"
2030 GOTO 1200
2035 REM
```

Subprogramas de instrucciones de máquina

```
2100 REM DETENCION
2105 RETURN
2200 REM INTRODUCCION
2205 PRINT "INTRODUCCION";
2210 INPUT N$
2215 REM VALIDACION
2220 IF LEN(N$) <> 2 THEN 2260
2225 LET H$ = MID$(N$, 1, 1)
2230 GOSUB 3900: REM CONVERSION A DECIMAL
2235 IF D = 16 THEN 2260
2240 LET H$ = MID$(N$, 2, 1)
2245 GOSUB 3900: REM CONVERSION A DECIMAL
2250 IF D = 16 THEN 2260
2255 GOTO 2270
2260 PRINT "INTRODUCCION INCORRECTA. POR FAVOR
           REPITALA";
2265 GOTO 2210
2270 LET A$ = N$
2275 RETURN
2300 REM SALIDA
2305 LET U$ = A$
2310 RETURN
2400 REM CARGAR
2405 LET E$ = D$
2410 GOSUB 3700: REM LECTURA DE LA MEMORIA
2415 LET A$ = G$
2420 RETURN
2500 REM ALMACENAR
2505 LET E$ = D$
2510 LET G$ = A$
2515 GOSUB 3800: REM ESCRITURA EN MEMORIA
2520 RETURN
2600 REM SUMAR
2605 LET E$ = D$
```

```
2610 GOSUB 3700: REM LECTURA DE LA MEMORIA
2615 LET H$ = MID$(A$, 1, 1)
2620 GOSUB 3900: REM CONVERSION A DECIMAL
2625 LET A = D
2630 LET H$ = MID$(A$, 2, 1)
2635 GOSUB 3900: REM CONVERSION A DECIMAL
2640 LET A = A  * 16 + D
2645 LET H$ = MID$(G$, 1, 1)
2650 GOSUB 3900: REM CONVERSION A DECIMAL
2655 LET G = D
2660 LET H$ = MID$(G$, 2, 1)
2665 GOSUB 3900: REM CONVERSION A DECIMAL
2670 LET G = G  * 16 + D
2675 LET A = A + G
2680 IF A < 256 THEN 2690
2685 LET A = A - 256
2690 LET D = INT(A/16)
2695 GOSUB 4000: REM CONVERSION A HEXADECIMAL
2700 LET A$ = H$
2705 LET D = A - D * 16
2710 GOSUB 4000: REM CONVERSION A HEXADECIMAL
2715 LET A$ = A$ + H$
2720 RETURN
2750 REM BORRAR
2755 LET A$ = "00"
2760 RETURN
2800 REM INCREMENTAR
2805 REM SE DEBE ESCRIBIR COMO EJERCICIO
2895 RETURN
2900 REM NEGAR
2905 LET H$ = MID$(A$, 1, 1)
2910 GOSUB 3900: REM CONVERSION A DECIMAL
2915 LET A = D
2920 LET H$ = MID$(A$, 1, 2)
2925 GOSUB 3900: REM CONVERSION A DECIMAL
2930 LET A = A * 16 + D
2935 LET A = 255 - A
2940 LET D = INT(A/16)
2945 GOSUB 4000: REM CONVERSION A HEXADECIMAL
2950 LET A$ = H$
2955 LET D = A - D * 16
2960 GOSUB 4000: REM CONVERSION A HEXADECIMAL
2965 LET A$ = A$ + H$
2970 RETURN
3000 REM BIFURCAR (BRN)
3005 LET C$ = D$
3010 RETURN
3100 REM BIFURCAR (BZE)
3105 REM SE DEBE ESCRIBIR COMO EJERCICIO
3110 RETURN
3200 REM BIFURCAR (BGT)
3205 REM SE DEBE ESCRIBIR COMO EJERCICIO
3210 RETURN
3300 REM BIFURCAR (BNG)
```

```
3305 REM SE DEBE ESCRIBIR COMO EJERCICIO
3310 RETURN
3400 REM OPERACION LIBRE
3405 RETURN
3500 REM OPERACION LIBRE
3505 RETURN
3600 REM OPERACION LIBRE
3605 RETURN
```

Leer en la memoria central

```
3700 REM LECTURA DE LA MEMORIA
3705 LET H$ = E$
3710 GOSUB 2900: REM CONVERSION A DECIMAL
3715 LET G$ = S$(D)
3720 RETURN
```

Escribir en la memoria central

```
3800 REM ESCRITURA EN MEMORIA
3805 LET H$ = E$
3810 GOSUB 2900: REM CONVERSION A DECIMAL
3815 LET S$(D) = G$
3820 RETURN
```

Conversión de hexadecimal a decimal

```
3900 REM CONVERSION DE HEXADECIMAL A DECIMAL
3905 REM PARAMETROS
3910 REM H$: DIGITO HEXADECIMAL
3915 REM D: EQUIVALENTE DECIMAL
3920 LET D = 16: REM INDICA ERROR
3925 FOR J = 0 TO 15
3930 IF H$ <> X$(K) THEN 3945
3935 LET D = J
3940 LET J = 15
3945 NEXT J
3950 RETURN
```

Conversión de decimal a hexadecimal

```
4000 REM CONVERSION DE DIGITO DECIMAL A HEXADECI
         MAL
4005 REM PARAMETROS
4010 REM H$: DIGITO HEXADECIMAL
4015 REM D: EQUIVALENTE DECIMAL
4020 LET H$ = X$(D)
4025 RETURN
```

Módulo PARAR

```
9900 REM MODULO DE PARAR
9905 PRINT
9910 PRINT "FIN DEL SIMULADOR MOS"
9915 END
```

Puntos de interés

- La suma de dos números hexadecimales (líneas 2600 a 2720) es sorprendentemente complicada. Para realizarla, se convierten los números a base decimal, se suman, y vuelven a transformarse a base hexadecimal.
- Si en cualquier fase un registro excede de su valor límite (15 para registros de 4 bits, 255 para los de 8 bits) se vuelve a cero y empieza de nuevo a incrementarse a partir de ahí. Esto equivale a ignorar todos los bits superiores a uno determinado.
- El módulo encargado de convertir números hexadecimales en decimales se ha aprovechado también para convalidar los primeros en el momento de su entrada.
- En varios puntos del programa se ha dejado espacio libre para facilitar futuras ampliaciones. El módulo de mando cuenta con sitio suficiente para órdenes adicionales y, además, se han escrito subprogramas en blanco para alojar en ellos las tres operaciones que aún quedan libres.
- En la modalidad de funcionamiento paso a paso es preciso introducir un carácter al término de cada instrucción para que comience el ciclo de la siguiente. Para interrumpir el programa se introduce por el teclado la palabra **INTERRUMPIR** en el momento en que se solicite.

29.7 Lenguaje ensamblador

El modelo de un ordenador debe contar necesariamente con un lenguaje máquina, pero, para ilustrar cabalmente su funcionamiento, es conveniente que además disponga de un **lenguaje ensamblador.** Programar en este lenguaje es algo más fácil que hacerlo en el de máquina. Sus características son las siguientes:

- A cada instrucción de máquina corresponde otra de lenguaje ensamblador identificada por medio de una **palabra simbólica** o un **mnemónico.** Esto significa que hay una correspondencia biunívoca entre las instrucciones dadas en lenguaje máquina y las dadas en lenguaje ensamblador.

- Las direcciones suelen indicarse mediante **etiquetas**, que pueden ser combinaciones de letras y cifras decimales.
- En este lenguaje, los números pueden escribirse en base decimal.
- El ensamblador contiene varias **seudoinstrucciones** que no corresponden a ninguna instrucción de lenguaje máquina.

Teniendo en cuenta estos puntos, el lenguaje ensamblador del MOS se define como sigue:

- Las operaciones se identifican por medio de una palabra simbólica de tres letras, como ilustra la figura 29.2.
- Las direcciones se representan mediante una etiqueta con una sola letra, que puede colocarse al principio de la línea para identificarla o tras la palabra simbólica de instrucción para identificar la dirección a que se hace referencia en la misma.
- Los números entran y salen en forma de enteros decimales comprendidos entre — 128 y 127.
- Hay dos seudoinstrucciones:

  DTA reservar la posición de memoria para un dato.
  FIN fin del programa.

29.8

Ejemplo de programa escrito en el ensamblador del MOS

El programa ejemplo de la sección 29.4 queda de la siguiente manera escrito en el ensamblador del MOS:

Etiqueta	*Instrucción*	*Interpretación*
	INT	Introducir el primer número.
	ALM X	Almacenarlo en la posición **X**.
	INT	Introducir el segundo número.
	NEG	Cambiar el signo del acumulador.
	INC	Sumar 1 al acumulador (se obtiene así el complemento del segundo número).
	SUM X	Sumar el primer número (se obtiene así la diferencia entre los dos).
	SAL	Llevar el resultado a la salida.
	DET	Detenerse.
X	**DTA**	Posición de memoria para el primer número.
	FIN	Fin del programa.

Observe que la etiqueta de la penúltima línea es igual a las de las dos instrucciones.

29.9

Programa ensamblador

Para aceptar programas en lenguaje ensamblador y traducirlos a lenguaje máquina es preciso ampliar el programa de simulación para incluir en él un **programa ensamblador.** La forma más fácil de programa ensamblador es la de **dos pasos,** llamada así porque recorre dos veces el programa escrito en lenguaje ensamblador.

En la primera pasada:

- Todos los contenidos de la columna de etiquetas se almacenan en una tabla junto con las direcciones a que se refieren las instrucciones o seudoinstrucciones correspondientes.
- Se verifican las palabras simbólicas y se transforman en el código de operación de la instrucción máquina que les corresponde.
- Se verifican las seudoinstrucciones **DTA** y se les asignan posiciones de memoria (llenas de ceros).

Si se detecta algún error, se presenta un mensaje y se interrumpe el proceso de compilación.

En la segunda pasada:

- Se buscan en la tabla las etiquetas contenidas en las instrucciones y se les asignan las direcciones correspondientes.

El programa en lenguaje máquina resultante puede almacenarse directamente en posiciones de la memoria central o acumularse primero en una matriz que a continuación se carga en la memoria.

Con esta breve descripción y algunos conocimientos básicos sobre lenguajes ensambladores, cualquiera debería ser capaz de escribir un programa ensamblador para el programa de simulación del MOS.

29.10

Conclusión

Este capítulo ha proporcionado una noción muy general de los rasgos esenciales de la estructura de la unidad de proceso y de los lenguajes máquina y ensamblador y ha enseñado a materializar todos esos conceptos en un modelo de ordenador. Este —bautizado MOS— se ha discutido en términos de la disposición de los registros y las características de sus lenguajes máquina y ensamblador. Ade-

más, se ha desarrollado un programa que simula ese lenguaje máquina y utiliza los registros del MOS. Los puntos de más importancia son:

- Un modelo de ordenador debe satisfacer ante todo las siguientes exigencias: responder a los conceptos generales de estructura de la unidad de proceso de la forma habitual en la práctica y hacerlo de forma adecuada al nivel del curso en que pretende utilizarse el modelo.
- El programa de simulación sirve para hacer una demostración sencilla y eficaz del funcionamiento del modelo de un ordenador.

29

Ejercicio

1. Defina brevemente los siguientes términos o expresiones: emulación, simulación, modelo de ordenador, registro, canal *(bus)*, lenguaje máquina, ciclo de instrucción, dirección, acumulador, contador del programa, registro de instrucción. código de operación. ciclo de memoria. lenguaje ensamblador. término mnemotécnico. etiqueta. seudoinstrucción. programa ensamblador.
2. Complete las partes que faltan del programa ejemplo 29.1:
   a) Las instrucciones iniciales para el usuario.
   b) El módulo **VISUALIZAR**.
   c) Los módulos encargados de ejecutar las instrucciones **INC, BZE, BGT** y **BNG.**
   Por lo que respecta a las instrucciones que implican el uso de números negativos. recuerde que éstos se reconocen por el valor 1 del bit más significativo, de forma que la primera cifra hexadecimal del registro es 8 o más.
   d) Amplíe el programa de simulación incorporándole un programa ensamblador que acepte como entrada un programa escrito en el lenguaje ensamblador del MOS. lo compile y cargue el código de máquina equivalente en las posiciones de memoria del MOS. partiendo de la dirección cero.
3. Escriba en lenguaje máquina o ensamblador MOS —o en los dos— programas que ejecuten algunas de las tareas siguientes o todas ellas:
   a) Introducir una serie de caracteres numéricos de cualquier longitud terminada en cero y presentar a la salida la suma de todos los números. Salvo que el programa haya sido modificado. los números han de introducirse en forma hexadecimal.
   b) Introducir dos números y llevar a la salida el mayor de ellos.
   c) Introducir dos números y multiplicarlos por un proceso de adiciones repetidas. Utilice el programa de simulación o una versión modificada del mismo para pasar los programas.
4. Modifique el programa de simulación del MOS para que presente en pantalla una imagen similar a la figura 29.1 como respuesta a la orden **VISUALIZAR** y también durante el pase de un programa. pero en esta ocasión con valores de registros y posiciones de memoria que cambien a medida que vayan ejecutándose las instrucciones.

5. Elija instrucciones adecuadas para ocupar los códigos de operación libres del lenguaje máquina del MOS. He aquí algunas sugerencias:

- una instrucción de substracción;
- una operación de álgebra de Boole. como Y u O;
- una operación de desplazamiento.

Escriba módulos para todas las instrucciones, haciendo uso, cuando sea necesario, de los módulos de servicio existentes.

6. Redacte la documentación para el programador y la guía para el usuario correspondientes al programa de simulación del MOS o a una versión modificada del mismo.
7. Diseñe un modelo de ordenador de su propia invención. Tenga en cuenta durante el proceso de diseño los puntos que se citarán a continuación y no empiece a escribir el programa hasta no haber planificado muy meticulosamente el diseño:

- Determine los objetivos del modelo (servir de elemento de enseñanza. poner a prueba una idea nueva referente a la estructura de la unidad de proceso. etc.).
- Determine la **longitud de palabra** del modelo, que es el número de bits de un dato o instrucción de máquina. Determina el número de bits de casi todos los registros.
- Determine el número de bits de una dirección. Para mayor simplicidad. puede hacerse igual a la longitud de palabra.
- Determine los registros que tendrá la unidad de proceso. Algunas disponen de varios acumuladores y de registros especiales (de índices o de indicadores de pila. por ejemplo). Otras cuentan con un banco de registros de tipo general que pueden usarse como acumuladores. como registros de índices o de indicadores. etc.
- Determine las **modalidades de direccionamiento** del ordenador simulado. El MOS sólo tiene uno. llamado direccionamiento **directo** o **absoluto.** Otras modalidades son: indirecto. indexado. relativo. inmediato y combinaciones de éstas.
- Determine el formato de una instrucción de máquina. Puede constar de un código de operación y una dirección en una sola palabra. o de un código de operación contenido en una palabra y una dirección en la siguiente. El código de operación debe disponer de campos para el tipo de instrucción. el registro y la modalidad de direccionamiento.
- Dibuje un diagrama que ilustre la disposición de los registros. la memoria central y los canales *(buses)* con el mismo detalle que la figura 29.1.
- Diseñe un programa de simulación del modelo que tenga la misma estructura general del programa ejemplo 29.1.
- Y procure que el resultado sea lo más sencillo posible.

8. Además de simular unidades de proceso de tipo general. como el MOS. pueden escribirse programas que ilustren el funcionamiento de determinados aspectos estructurales. como la conexión en serie. el proceso en paralelo o el contenido direccionable de la memoria (para más detalles sobre estos aspectos. véase *Introducción a la ciencia de la informática (An introduction to Computing Science)*, capítulo 11).


RESET
RETURN

30
Propuestas de trabajos

Uno de los objetivos de cualquier curso avanzado de programación es conseguir que quienes lo sigan sean capaces de escribir programas serios de calidad aceptable. Por lo general, esa capacidad se valora por medio de un trabajo, que suele incluir un programa o una serie de ellos.

En este capítulo, que cierra el libro, haremos algunas sugerencias útiles para los que deseen planificar un trabajo como el mencionado. Daremos primero unas normas generales, para a continuación pasar a proponer una "lista de compras" con trabajos sobre temas específicos. Por otra parte, en muchos capítulos de este libro, y sobre todo en los que van del 22 al 29, dedicados al estudio de aplicaciones de la informática, se han dado ya algunas ideas a este respecto.

Cada uno de los trabajos propuestos empieza por una breve presentación a la que, casi siempre, sigue una exposición de los principios que sustentan el tema de estudio. Muchas de las propuestas de estudio realizadas se refieren a temas que son a su vez interesantes motivos de estudio. Todo lo que podemos hacer aquí es proporcionar suficiente información para que la persona interesada esté en condiciones de decidir si continúa o no la investigación. El material aportado aquí no llega en ningún caso a constituir una base suficiente sobre la que elaborar un trabajo. Al final del capítulo se da una lista de posibles fuentes de información adicional.

Título del trabajo	*Disciplina*
1. Conjunto estadístico	Estadística
2. Manipulación de archivos	Tratamiento de datos
3. Banco de datos	Tratamiento de datos
4. Conducción de calor	Física y química
5. Balística	Física
6. Nudos ferroviarios	Matemáticas aplicadas
7. Música	Música
8. Modelos económicos	Economía y estadística
9. Genética	Biología
10. Ecología	Biología y geografía

30.1

Algunas consideraciones generales

Al planificar un trabajo para un nivel dado, es importante tener en cuenta una serie de normas generales, normas que no deben considerarse reglas rígidas, sino ideas que han de interpretarse de forma flexible y acorde con la situación.

El fallo más común de los trabajos a este nivel es que son demasiado generales y tratan de abarcar demasiado. Al planificar un trabajo es importante:

- determinar el fondo ante el que opera, aunque sea completamente imaginario;
- definir los objetivos del trabajo y limitarlos a un nivel razonable.

Una vez determinado el fondo y fijados los objetivos generales, hay que localizar un sistema del que formen parte los programas. Que el resto del sistema que sirve de fondo no sea utilizado por el programa o que sea completamente ficticio son cosas que no tienen importancia, porque un programa de ordenador no tiene naturaleza autónoma. Por lo general, el sistema se define mediante una serie de suposiciones o hipótesis.

Una vez perfilado el sistema que envuelve al programa, hay que pasar a ocuparse de éste. Es importante establecer los objetivos del mismo, que son diferentes de los del trabajo considerado globalmente. Un diseño sistemático, como el descrito y utilizado en este libro, es algo imprescindible; el programa no debe escribirse hasta después de terminado y comprobado el diseño.

El programa debe estructurarse de forma que puedan introducirse correcciones y modificaciones sin necesidad de alterarlo en lo esencial.

En ocasiones son útiles los diagramas de flujo, pero nunca resultan imprescindibles. Pueden elaborarse a dos niveles: a nivel del sistema para ilustrar su estructura general, y a nivel del programa, como se ha hecho en este libro. En cualquier caso, los diagramas deben incluirse si contribuyen a aclarar conceptos o técnicas o si facilitan el proceso de diseño del programa; pero si ocupan mucho espacio y no aportan gran cosa, deben omitirse.

Hay que preparar un lote generoso de datos de prueba para verificar los módulos del programa, o el programa completo, bajo diferentes condiciones. Hay que incluir en ellos tanto lo típico como lo extremo, más algunos datos pensados especialmente para intentar que el programa falle.

La documentación es importantísima y, de hecho, constituye el grueso de casi todos los trabajos. Además de la documentación para el programador y la guía del usuario, hay que incluir en este apartado todas las notas que forman parte del programa y mejoran su legibilidad. La calidad del lenguaje de la documentación es de importancia capital: hay que usar siempre un estilo claro y conciso.

En muchos casos viene bien proceder a una evaluación interna del programa para localizar los aspectos susceptibles de mejora dentro del tiempo disponible para realizar el trabajo. Y aun si tal cosa es imposible, no tiene nada de malo ser consciente de las virtudes y los puntos flacos del soporte lógico: al fin y al cabo, ningún programa es perfecto.

En resumen:

- sea breve;
- no complique las cosas innecesariamente;
- aténgase a la fecha de entrega.

30.2 Propuesta 1: Paquete estadístico

Un programa capaz de calcular diversos parámetros estadísticos a partir de uno o más conjuntos de datos observados tendrá siempre mucha utilidad. A continuación se sugieren algunos de los cálculos que podría ejecutar.

Media

La media $\bar{x}$ se determina mediante la fórmula

$$\bar{x} = \frac{1}{n}\sum x$$

donde $\sum x$ es la suma de todos los valores de x y n es el número de valores sumados.

Desviación típica

La desviación típica S de un conjunto de datos se determina mediante una de las siguientes fórmulas:

$$S^2 = \frac{\sum x^2}{n} - \left(\frac{\sum x}{n}\right)^2$$

o

$$S^2 = \frac{\sum x^2}{n} - (\bar{x})^2$$

Coeficiente de correlación

El coeficiente de correción R de dos conjuntos de datos se determina mediante la fórmula

$$R = \frac{(\sum xy - n\overline{xy})}{\sqrt{(\sum x^2 - n\bar{x}^2)(\sum y^2 - n\bar{y}^2)}}$$

Ajuste de la curva de mínimos cuadrados

Si los valores de los dos conjuntos de datos se han representado como puntos, la recta que mejor se ajusta a dichos puntos se determina como sigue:

Siendo la ecuación de la línea

$$y = ax + b$$

los parámetros a y b se obtienen con las fórmulas

$$a = \frac{\sum xy - n\overline{xy}}{\sum x^2 - n\bar{\bar{x}}^2}$$

y

$$b = \bar{y} - a\bar{x}$$

Otras posibilidades

Puede ampliarse el programa para que calcule la mediana de un conjunto de datos (se obtiene clasificando los datos en orden y escogiendo el valor medio) y realice tratamientos más complejos, como el análisis de varianza, la presentación de los datos organizados en tablas, histogramas o curvas, la edición de conjuntos de datos introducidos con anterioridad o el almacenamiento y la recuperación de datos almacenados en ficheros.

30.3 Propuesta 2: Manipulación de archivos

Gran número de sistemas de tratamiento de datos comerciales y administrativos son básicamente aplicaciones de manipulación de ficheros, que implican el almacenamiento y la actualización de grandes ficheros y el tratamiento de los datos que contienen.

Si se piensa en un trabajo sobre manipulación de ficheros, es imprescindible basarlo en una aplicación particular, porque un sistema de esa clase de tipo general no sirve de nada. Son aplicaciones generales de esta clase los sistemas de pago de nóminas, de control de existencias, de contabilidad, etc., pero aun dentro de estas opciones es preciso centrarse en aspectos más específicos.

Una vez decidida la aplicación, las fases generales del trabajo son:

- Determinar los objetivos del programa o conjunto de programas de manipulación de ficheros.
- Diseñar los ficheros, es decir, determinar su contenido y su estructura.
- Determinar las operaciones que han de ejecutar los programas de manipulación, que probablemente serán:

  introducción de datos;
  validación;
  almacenamiento de los datos válidos en un fichero;
  edición de los datos de un fichero;
  clasificación de los registros de un fichero;
  actualización de un fichero, por lo general comparando un fichero maestro con otro de operaciones para obtener un nuevo maestro;
  impresión de todos los datos de un fichero;
  obtención de un informe basado en el contenido del fichero;
  búsqueda en un fichero de un registro específico;
  hacer una copia auxiliar del fichero, es decir, reproducirlo en otro medio de almacenamiento;
  eliminar un fichero.

- Es aconsejable incluir un mecanismo de seguridad con una contraseña que franquee el acceso al fichero.

Aunque en la práctica eso no se cumple casi nunca, conviene que los ficheros sean suficientemente cortos como para caber en la memoria central del ordenador.

30.4

Propuesta 3: Banco de datos

Se trata de un ejercicio atractivo y difícil, ya que implica a la vez tratamiento de ficheros y análisis sintáctico. Un banco de datos es un sistema que mantiene un gran archivo de información y extrae subconjuntos del mismo en respuesta a las solicitudes del usuario.

Un ejemplo de esto sería un fichero de alumnos de una escuela, con datos ordenados bajo los encabezamientos

Apellido   *Nombre*   *Año*   *Grupo*   *Otros*

La solicitud de información podría adoptar la forma

Año = 3 Y francés en Otros
Imprimir Apellido, Nombre, Grupo

En este caso, se imprimirá el apellido, el nombre y el grupo de todos los alumnos de tercer año que estudien francés.

Dado que elaborar un programa de esta naturaleza es más difícil de lo que parece, no hay más remedio que elegir una aplicación sencilla y muy específica y adaptar el programa a la información archivada. Las tareas que deberá realizar son:

- Actualizar, editar y ampliar la información del fichero.
- Aceptar preguntas formuladas con arreglo a un patrón fijo. Las preguntas se formulan en un lenguaje sencillo e implican condiciones quizá relacionadas con operaciones del álgebra de Boole.
- Verificar la sintaxis de las preguntas y presentar mensajes de error adecuados si están mal formuladas.
- Comparar las preguntas con los registros del fichero y extraer los campos de aquéllos que satisfagan las condiciones solicitadas.

Propuesta 4: Conducción de calor

Los sistemas de conducción de calor son numerosos y variados y de un tamaño que oscila entre el de una instalación de calefacción central doméstica y el de una planta siderúrgica. El continuo incremento del coste de la energía hace que el conocimiento profundo de tales sistemas sea cada vez más importante.

Un trabajo de simulación de un sistema de flujo térmico combina elementos de física, de matemáticas aplicadas y de trazado de curvas. El objetivo de la simulación es estudiar el comportamiento del sistema bajo diferentes condiciones y determinar el efecto sobre el mismo de varias formas de control.

Un ejemplo sencillo ilustrará cabalmente los conceptos en juego. La figura 30.1 representa un depósito doméstico de agua calentado por una resistencia controlada por un termostato. Con las variables del dibujo, el cambio en la situación durante un período de tiempo breve Δt se expresa como sigue:

Ganancia de calor = potencia de la resistencia × tiempo = $P\Delta t$

Pérdida de calor = superficie × conductividad × diferencia de temperatura × tiempo =

$$= AC(T_1 - T_2)\Delta t$$

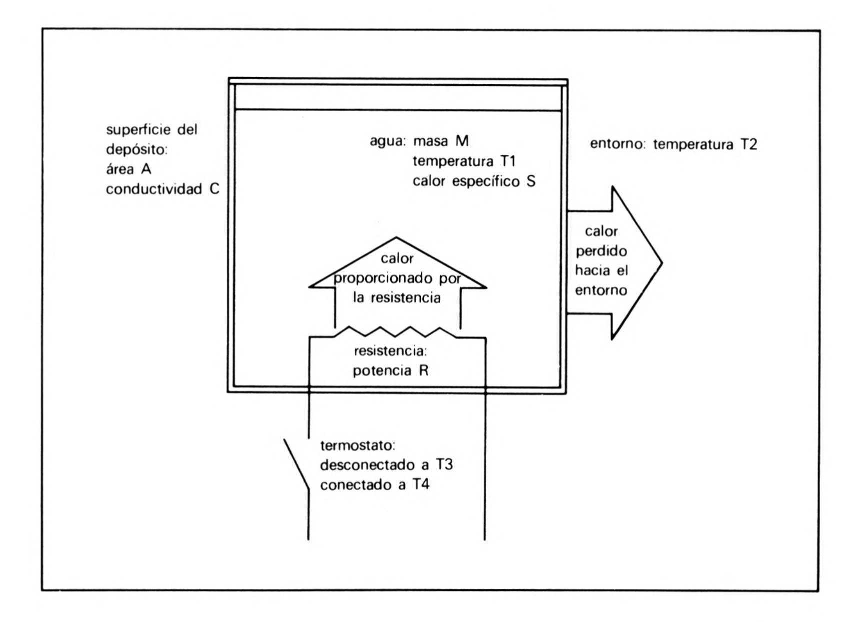


Figura 30.1
Depósito de agua caliente

$$\text{Ganancia de calor neta} = R\Delta t - AC(T_1 - T_2)\Delta t$$

$$\text{Incremento de temperatura} = \frac{\text{ganancia de calor neta}}{\text{masa} \times \text{calor específico}} =$$

$$= \frac{(R - AC(T_1 - T_2))\Delta t}{MS}$$

$$\text{Nueva temperatura} = T_1 + \text{incremento de temperatura.}$$

Las ecuaciones propuestas pueden usarse de forma repetitiva para simular el comportamiento del sistema a lo largo de cierto período de tiempo. Es razonable suponer que la temperatura del entorno T_2 permanece constante durante dicho período.

La simulación puede utilizarse para examinar el efecto de diversas posiciones del termostato y para calcular la necesidad total de energía del sistema, que es igual al producto de la potencia del elemento calefactor por el tiempo durante el que está conectado.

Con lo dicho hasta ahora pueden simularse situaciones muy variadas, pero conviene no abarcar demasiado. El estudio de sistemas calentados total o parcialmente por reacciones químicas también es posible, pero resulta considerablemente más complicado.

La mejor forma de presentar los resultados a la salida es un esquema animado del sistema en el que las cifras de temperatura vayan cambiando con el tiempo. Si se dispone de un equipo en color, éste puede modificarse en función de las temperaturas. Caso de que resulte imposible esta opción del esquema animado, lo más aconsejable es una curva de tiempos y temperaturas.

Nota sobre las unidades

En las ecuaciones debe emplearse un conjunto coherente de unidades; el sistema más recomendable es el MKS:

Calor: julios.

Potencia (del elemento calefactor): watios o julios/segundo.

Tiempo: segundos.

Superficie: metros cuadrados.

Masa: kilogramos.

Temperatura: grados centígrados.

Conductividad térmica: julios/metro cuadrado/segundo.

Calor específico: julios/kilogramo/grado centígrado.

30.6

Propuesta 5: Balística

Una de las primeras aplicaciones de los ordenadores fue el cálculo de las trayectorias de proyectiles, sobre todo de granadas de artillería y bombas lanzadas desde aviones. Tales cálculos siguen siendo interesantes, sobre todo si se combinan con esquemas gráficos.

Aunque, estrictamente hablando, un proyectil es un objeto que se lanza y que carece de fuerza impulsora propia, el estudio puede ampliarse para incluir cohetes utilizando las técnicas de aproximación propuestas aquí.

La trayectoria de un proyectil suele estudiarse en términos de dos dimensiones: la altitud y la distancia horizontal recorrida. La formulación matemática del problema se simplifica notablemente si las dos se tratan por separado. Lo importante es la posición, la velocidad y la aceleración del proyectil en un instante dado y el ángulo que su trayectoria forma con la horizontal en ese instante. La figura 30.2 ilustra una trayectoria típica y recoge algunas de las variables necesarias para describirla en términos matemáticos.

Las ecuaciones que describen la trayectoria de un proyectil se basan en la **ley fundamental del movimiento de Newton.** Para un proyectil de masa *m*, impulsado por un cohete que desarrolla

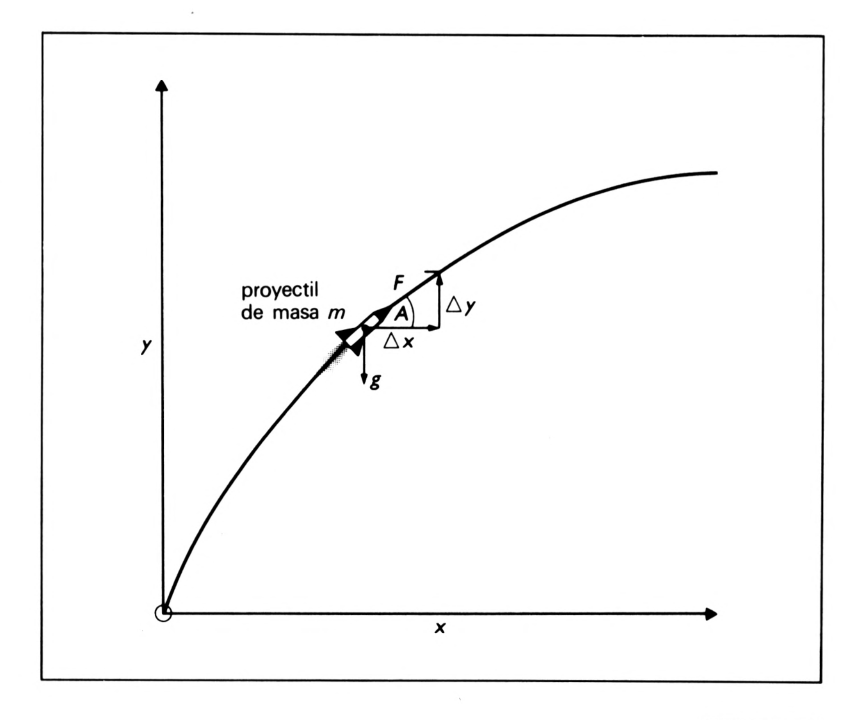


Figura 30.2
Movimiento de un proyectil

un empuje F y se desplaza con un ángulo respecto a la horizontal A, se tiene:

$$\text{Aceleración vertical } a_y = \frac{1}{m}(F \text{ sen } A - mg)$$

$$\text{Aceleración horizontal } a_x = \frac{1}{m}(F \cos A)$$

La aceleración representa el ritmo a que cambia una velocidad. Por tanto, a lo largo de un intervalo de tiempo breve Δt,

cambio de velocidad vertical $\Delta v_y = a_y \Delta t$
cambio de velocidad horizontal $\Delta v_x = a_x \Delta t$

La velocidad representa a su vez el ritmo de cambio de posición. Por tanto, durante el intervalo de tiempo Δt,

cambio de posición vertical $\Delta y = v_y \Delta t$
cambio de posición horizontal $\Delta x = v_x \Delta t$

Cada uno de los cambios indicados se suma al valor anterior de la variable para obtener el valor nuevo de la misma. El nuevo ángulo de inclinación se calcula mediante la fórmula

$$\tan A = \frac{\Delta y}{\Delta x}$$

Estas ecuaciones pueden usarse iterativamente a partir de unos valores iniciales adecuados para calcular la trayectoria de un proyectil en forma de un conjunto de coordenadas verticales y horizontales. El empuje puede variar a lo largo del tiempo, lo mismo que la masa en el caso de un cohete (ya que el combustible se quema). De la misma forma, es posible ampliar las ecuaciones para tener en cuenta la resistencia del aire.

La mejor forma de presentación de los resultados es en tiempo real, quizá elaborando un gráfico móvil de la trayectoria. También pueden elaborarse curvas de varias magnitudes o tablas. Se recomienda trabajar en el sistema de unidades MKS.

30.7

Propuesta 6: Señalización de un nudo ferroviario

Un trabajo interesante y con aplicaciones prácticas es la simulación del funcionamiento de un cruce de vías y de la señalización del mismo. Veamos un ejemplo de esta propuesta.

La figura 30.3 ilustra la disposición de las agujas y las señales en un cruce sencillo y las zonas adyacentes al mismo. El cruce se simula mediante intervalos de tiempo suficientemente largos para que el tren pueda pasar de una zona a la siguiente. Durante cada uno de dichos intervalos:

- Pueden introducirse códigos de tres en las zonas T_1, T_2 y T_3, siempre que la zona en cuestión esté libre.
- Las agujas se colocan con arreglo al código del tren que ocupa la zona T_1 y a las prioridades relativas de los que ocupan las T_2 y T_3.
- Los trenes situados en las zonas T_4, T_5 y T_6 pueden salir del sistema.

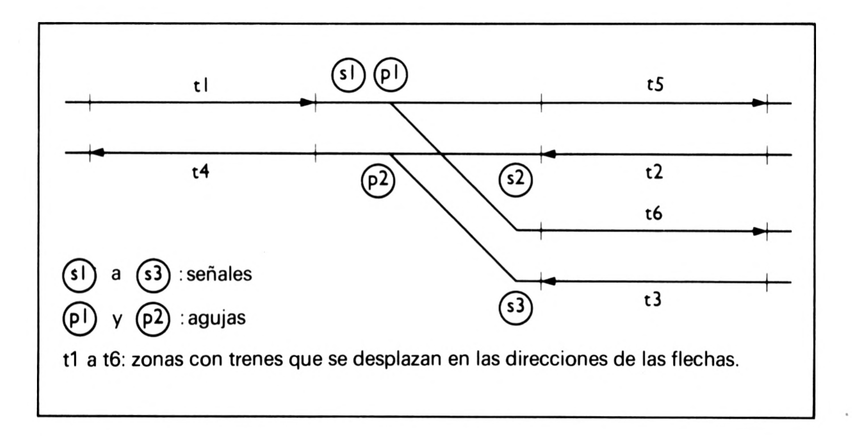


Figura 30.3
Un cruce de vías

- Las señales se colocan en verde según la posición de las agujas y si la zona situada por delante del tren está libre.
- Los trenes situados en las zonas T_1 a T_3 que se acerquen a una señal verde pueden avanzar hasta la zona siguiente.

Este es un ejemplo de simulación por **secuencia de acontecimientos,** por oposición a la simulación por intervalos de tiempo utilizada en otros ejemplos. Al planificar este trabajo es importante elegir una situación realista pero no demasiado compleja, a ser posible basada en un cruce real. Pueden investigarse las consecuencias de diferentes normas de señalización y distintas intensidades de tráfico.

La forma de salida recomendada es un gráfico similar al de la figura 30.3 que recoja las posiciones de las agujas, los colores de las señales y los códigos de los trenes que ocupan las diversas zonas. El esquema se actualiza tras cada movimiento de un tren.

30.8

Propuesta 7: Que no pare la música

Interpretada por un estilófono o por un sintetizador, la música generada por ordenador asciende con fuerza en las listas. Para componer con ordenador no hace falta ser un experto en teoría musical ni siquiera saber leer música, pero las dos cosas ayudan.

La idea básica consiste en describir para un ordenador la estructura fundamental de una pieza, especificando el ritmo, la clave, la secuencia de acordes y la duración, dejando a cargo de un programa la composición y, quizá, incluso la ejecución del resultado. Este puede obtenerse en la notación musical convencional (o en una buena aproximación a la misma) o en cualquier otra adecuada para el instrumento en uso. Si éste es un sintetizador o un instrumento de su naturaleza conectado directamente al ordenador o si se cuenta con un generador de sonidos, la máquina podrá, además de componer la pieza, interpretarla.

Aunque estas cosas no afectan a la música más vanguardista, cualquier composición más o menos normal tiene tres dimensiones: ritmo, melodía y armonía, y el programa debe actuar sobre las tres.

El ritmo sigue *(The Beat Goes On)*

El ritmo de una pieza determina el énfasis relativo que se pone en cada una de las notas y el compás general de la composición. En la mayor parte de las piezas clásicas y modernas, el ritmo consiste en una serie de variaciones en torno a una estructura básica. Clásicamente esta estructura se representa por medio de una serie de acentos o golpes; así, el vals tiene un compás de tres golpes, que en muchísimas otras composiciones es de cuatro; la música actual, y sobre todo el jazz, tienen ritmos casi siempre más complejos o más sutiles, aunque basados habitualmente en un compás de cuatro o de ocho. La música disco al uso, por ejemplo, sigue un ritmo

—·—·—·—*

donde — es un sonido fuerte, · uno débil y * uno muy fuerte; la duración de los sonidos es siempre la misma.

Este ritmo constituye el entramado o estructura de la pieza, sobre el que el programa crea variaciones por medio del generador de números aleatorios; por ejemplo:

0*0*00**

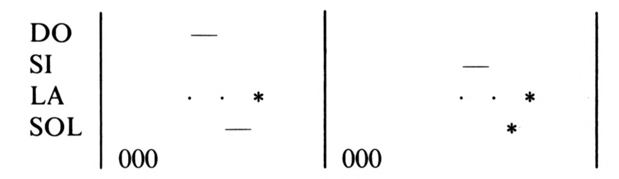
donde 0 es un compás de silencio.

El hacedor de melodías *(Melody maker)*

La melodía de una composición es su "sintonía" elemental, lo que de ella puede interpretarse con un solo instrumento o tararearse. Aunque la música electrónica se aleja cada vez más de la melodía única, ésta constituye un buen punto de partida.

Salvo que se trabaje con un sintetizador, que puede generar sonidos de cualquier frecuencia, la melodía se compone a partir de una serie de notas con una **clave** fija que define la **escala**; así, do mayor utiliza la escala natural do, re, mi, fa, sol, la, si. Una melodía debe mantenerse dentro de cierto intervalo de notas, que depende del instrumento en que vaya a interpretarse (éste puede también imponer limitaciones en cuanto a los saltos entre notas consecutivas u obligar a que las secuencias vayan en orden ascendente o descendente). El detalle de las notas se confía también al generador de números aleatorios.

En cuanto a la notación utilizada para la melodía, puede ser la habitual o alguna versión simplificada, como esta:



que corresponde a la secuencia de notas la do la sol la, la si la sol la; los 0 señalan silencios y la notación del ritmo es la que ya se ha explicado.

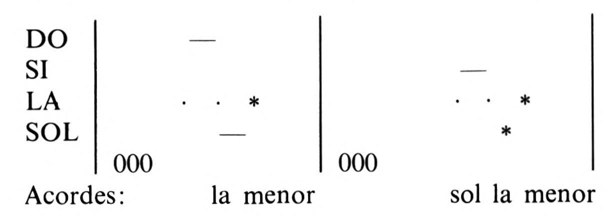
Vivir en armonía *(Living in harmony)*

Además de la melodía, la mayor parte de las composiciones incluyen una serie de acordes y notas bajas organizadas de forma que, en un momento dado, se ejecutan varias notas a la vez: se llama armonía (o falta de armonía) al efecto producido por el sonido combinado de esas notas.

Aunque la armonía es una disciplina muy compleja, daremos aquí unas pocas normas generales. El uso de **acordes** mayores y menores da lugar a dos efectos contrastantes. Un acorde mayor se caracteriza por un sonido brillante y alegre; es el caso de las notas do, mi y sol en acorde de do mayor. El acorde **menor** es más grave, como el producido por las notas la, do y mi en acorde de la menor. Por último, también puede emplearse deliberadamente la **disonancia** que producen ciertas notas, como do y fa cuando se ejecutan simultáneamente y con intensidad.

Las melodías más populares se basan en una **secuencia de acordes** con variaciones. Dicha secuencia puede especificarse como parte de la entrada del programa, pero también puede fijarla éste tras la composición de la melodía.

La forma de salida más sencilla de la armonía es indicar la nota que ha de ejecutarse en el punto preciso en que debe sonar. Así, serían acordes adecuados a la melodía anterior los siguientes:



Otros efectos

Si se trabaja con sintetizador o si se dispone de dispositivos especiales como desfasadores, ecos, cajas de ritmos, etc., las posibilidades aumentan muchísimo.

En cualquier caso, sea cual sea la técnica de programación y con independencia del método de salida y de los instrumentos que se utilicen, es preciso apañárselas para introducir en el ordenador un elemento vital: imaginación.

30.9

Propuesta 8: Modelos económicos

En economía se usan cada vez más los modelos informatizados, tanto para mejorar el conocimiento que se tiene sobre un área económica particular como para facilitar la toma de decisiones. Los modelos más ambiciosos tratan de simular el comportamiento del sistema económico completo, mientras que los menos ambiciosos investigan tendencias concretas o relaciones sectoriales. Si se elije un trabajo de esta clase, lo más aconsejable es limitarse al segundo tipo de simulación.

Los modelos económicos suelen proponer relaciones entre precios, salarios, niveles de desempleo y entrada de dinero y entre tipos de cambio (por lo general porcentajes de incremento) de esas cantidades. Se ponen a prueba recogiendo valores para las variables a lo largo de algunos años y utilizando técnicas estadísticas, como el ajuste de la curva de mínimos cuadrados, para calcular varios parámetros y establecer así la bondad del ajuste de las ecuaciones a

los hechos. Los modelos también pueden emplearse para prever las consecuencias de ciertas medidas de economía política, como la limitación del incremento de los salarios.

Esbozaremos a continuación dos modelos muy conocidos, no tanto porque su empleo sea el más recomendable cuanto porque permiten hacerse una idea de la clase de modelos utilizados habitualmente.

Ecuación de Fisher

El economista I. Fisher ha propuesto una ecuación que relaciona la cantidad de dinero en circulación (variable M) con el nivel medio de precios (variable P) en un sistema económico:

$$P = kM$$

El parámetro k depende de la velocidad de circulación del dinero y del flujo de bienes, y se supone constante a corto plazo.

El modelo puede ponerse a prueba utilizando cifras históricas de precios y dinero en circulación y haciendo estimaciones de k a lo largo de varios períodos. Partiendo del actual valor de ese parámetro pueden hacerse predicciones sobre las consecuencias de diversos cambios en la entrada de dinero.

Ecuación de Phillips

Esta ecuación, debida al economista A. Phillips, establece una relación entre el desempleo y el incremento salarial:

$$W = aU^b + c$$

donde W es el porcentaje anual de modificación de los salarios, U la tasa de desempleo y a, b y c, constantes con los siguientes valores:

$$a = 9{,}638$$
$$b = -1{,}394$$
$$c = -0{,}9$$

Para probar la verosimilitud del modelo hay que obtener cifras de tasas de desempleo y de incrementos salariales y, probablemente, también será necesario utilizar valores actualizados para las constantes. Estrictamente hablando, el modelo sólo debe utilizarse para prever futuras tendencias salariales a la vista de las cifras de desempleo, pero no al revés.

30.10

Propuesta 9: Genética

Los ordenadores cada vez se usan más en el estudio de la genética, ya que permiten simular las características de futuras generaciones de plantas o animales mucho más rápidamente que el proceso natural de cruzamiento y cría.

La genética clásica, basada en los descubrimientos de Gregorio Mendel, supone que cada una de las características de las plantas o los animales que se multiplican por reproducción sexual está determinado por dos **genes**, contenido uno de ellos en cada uno de los parentales. Dichos genes pueden ser **dominantes** o **recesivos;** dada una combinación de varios para una característica particular, el dominante será el que determine el aspecto del individuo, aunque tanto éste como el recesivo pasan a futuras generaciones.

La figura 30.4 ilustra el comportamiento de los genes dominantes y recesivos en unos de los célebres experimentos de Mendel con guisantes de flores rojas y blancas. Como el rojo es dominante, las plantas de la segunda generación tienen todas flores rojas. Pero como además se ha transmitido el gen blanco, en la tercera generación se forman 3 plantas de flores rojas por cada 1 de flores blancas.

Con experimentos similares puede investigarse la transmisión de combinaciones de varias características. Se ha demostrado que no todas las características son mutuamente independientes y que además ocurren mutaciones que dan lugar a la aparición de características nuevas.

La transmisión de los genes de los padres a los hijos es un proceso regido por el azar que, por tanto, se presta a ser simulado haciendo uso del generador de números aleatorios. El ordenador permite simular el funcionamiento de grupos muy amplios de individuos a lo largo de muchas generaciones, e incluso pueden incorporarse al modelo factores como el ligamiento de unas características con otras o la tasa de mutación. Los objetivos de la simulación son exponer el funcionamiento de la teoría genética e investigar el comportamiento de determinadas combinaciones de genes dominantes y recesivos a lo largo de las generaciones.

30.11

Propuesta 10: Un ecosistema

La ecología puede describirse como el estudio de los organismos en su medio ambiente. Se llama ecosistema a un medio identificable y susceptible de ser estudiado globalmente, como un lago o una región

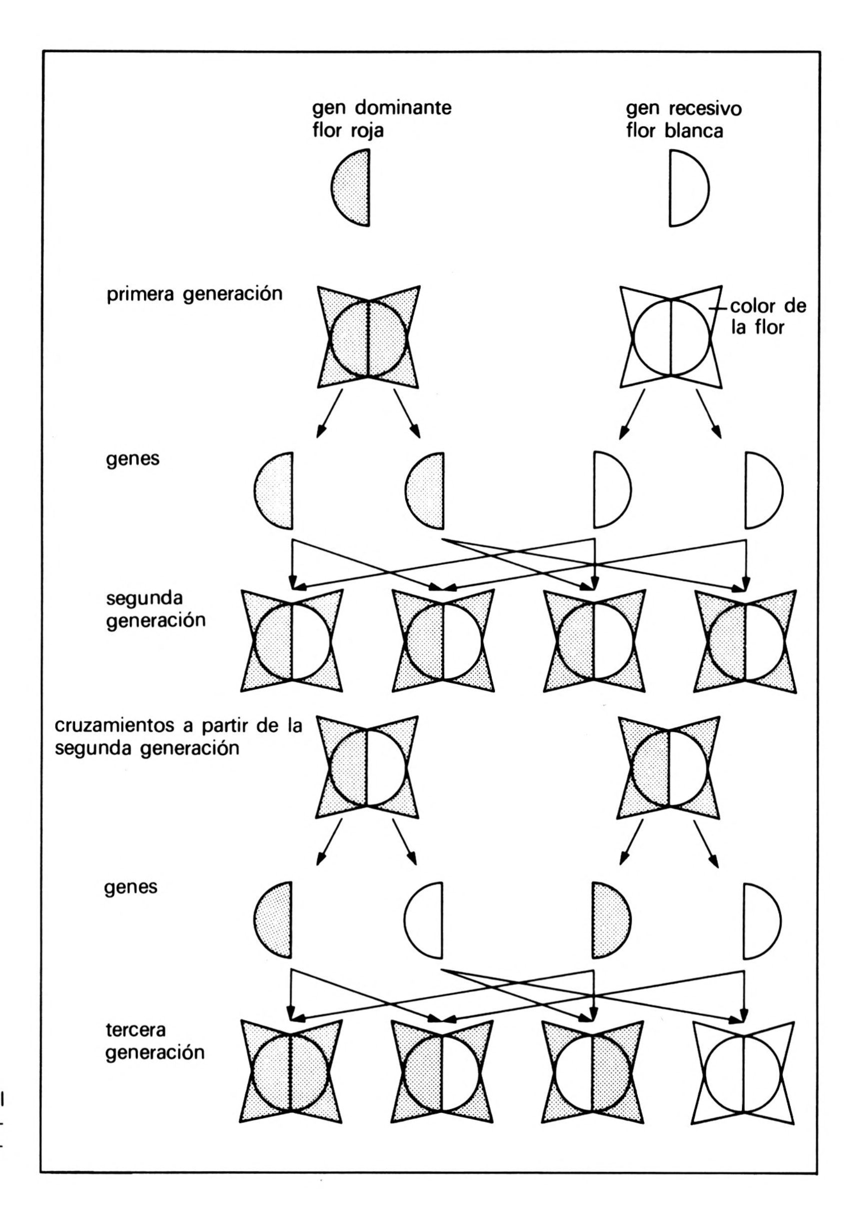


Figura 30.4
Experimento de Mendel sobre el comportamiento de los genes recesivos y dominantes en guisantes de flores rojas y blancas

con límites naturales. El creciente impacto de las sociedades industriales sobre su medio ha acrecentado considerablemente el interés del estudio de los ecosistemas.

La mayor dificultad del estudio de un ecosistema es su complejidad, ya que con frecuencia es necesario investigar cientos de especies diferentes que interaccionan de forma compleja y poco conocida. Justamente por eso es tan instructivo estudiar los ecosistemas por medio de modelos. Para establecer un modelo hay que proceder

sistemáticamente, simplificando el sistema sin sacrificar sus propiedades esenciales. Se han propuesto varios modelos diferentes hasta la fecha, y aquí describiremos uno de los más sencillos y más usados: el modelo de **compartimientos.**

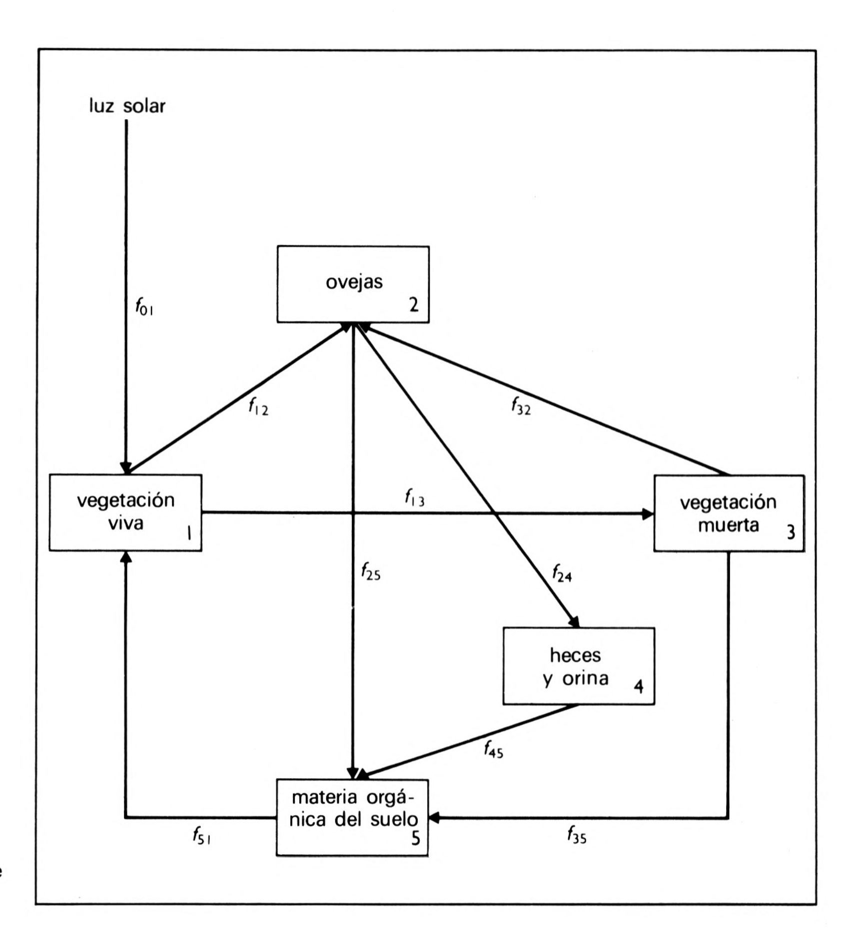


Figura 30.5
Modelo de compartimientos de un ecosistema

Este hace equivaler el ecosistema a un conjunto de compartimientos, cada uno de los cuales encierra una **cantidad,** sea la masa de una especie vegetal, el número de una población animal o un volumen de energía. Los compartimientos se unen por medio de **flujos** que representan el movimiento entre unos y otros; así, cuando un animal come hierba, se establece un flujo de masa del compartimiento de ésta al de aquél.

La figura 30.5 ilustra un modelo de compartimientos muy sencillo que podría utilizarse, por ejemplo, para describir el medio de una isla con una sola especie vegetal y poblada sólo por ovejas. Tiene varios

compartimientos (vegetación viva. vegetación muerta, ovejas, etc.) y una serie de flujos que corresponden a los siguientes intercambios:

f_{01} cantidad de luz solar (en unidades de energía) que recibe la vegetación viva en un intervalo de tiempo dado.

f_{12} masa de vegetación viva ingerida por las ovejas en un intervalo de tiempo dado.

f_{13} masa de vegetación viva que se seca en un intervalo de tiempo dado.

f_{24} masa de heces y orina excretada por las ovejas en un intervalo de tiempo dado.

f_{25} número (o masa) de ovejas muertas en un intervalo de tiempo dado.

f_{32} masa de vegetación muerta ingerida por las ovejas en un intervalo de tiempo dado.

f_{35} masa de vegetación muerta que se descompone en un intervalo de tiempo dado.

f_{45} masa de heces y orina que se descompone en un intervalo de tiempo dado.

f_{51} masa de materia orgánica incorporada por la vegetación viva en un intervalo de tiempo dado.

El modelo avanza en el tiempo calculando todos los flujos que se producen a partir de los niveles presentes en cada momento en cada uno de los compartimientos y a continuación determina la evolución de dichos niveles en función de los flujos. Casi siempre se utiliza un intervalo de tiempo bastante corto, como un día, una semana o un mes.

Cada flujo se determina mediante una función que depende de los niveles de los compartimientos de origen y destino. Los flujos que van del entorno exterior al sistema en estudio —la luz solar. por ejemplo— se calculan por separado. por lo general utilizando el generador de números aleatorios para calcular las variaciones a partir de una secuencia cíclica.

También se usa una función para determinar el nuevo nivel alcanzado por cada uno de los compartimientos a partir de todos los flujos que entran o salen de ellos. Hay que tener cuidado con las unidades, porque no todos los niveles y flujos se expresan en las mismas.

Un modelo de este tipo puede usarse para investigar la estabilidad de un ecosistema en un período de tiempo largo o las consecuencias de la caza y la pesca. las actividades agrícolas. el empleo de fertilizantes artificiales y plaguicidas o la introducción de contaminantes.

30.12

Fuentes adicionales de información

Damos a continuación los nombres y direcciones de algunas entidades y revistas que pueden proporcionar información adicional sobre diversos aspectos de la informática:

REVISTAS ESPECIALIZADAS

- EL ORDENADOR PERSONAL
  Ferraz, 11
  28008 MADRID

- MICROS / CHIP
  Ediciones Arcadia
  Víctor de la Serna, 4
  28016 MADRID

- TU MICRO
  Avenida de Alfonso XIII, 141
  28016 MADRID

- INFORMATICA TEST
  Haymarket, S. A.
  Travesera de Gracia, 17-21
  08021 BARCELONA

- ZX / TODOSPECTRUM / ORDENADOR POPULAR / COMMODORE MAGAZINE / PC MAGAZINE
  Bravo Murillo, 377
  Apartado de Correos 784
  MADRID

- MICROHOBBY
  Hobbypress
  Polígono Industrial de Alcobendas
  La Granja, 8
  Alcobendas
  MADRID

ESCUELAS Y FACULTADES

- E.T.S.I. Telecomunicaciones (EURIELEC: Club Técnico de Alumnos)
  Ciudad Universitaria, s/n
  28040 MADRID

- ESCUELA UNIVERSITARIA DE INFORMATICA
  Carretera de Valencia, Km 7
  MADRID

- FACULTAD DE INFORMATICA
  Carretera de Valencia, Km 7
  MADRID

- ESCUELA TECNICA DE TELECOMUNICACIONES
  Carretera de Valencia, Km 7
  MADRID

DISTRIBUIDORES

- MICPE (Apple)
  Valencia, 87
  08029 BARCELONA

- INVESTRONICA
  Tomás Bretón, 60
  28007 MADRID

- INDESCOMP
  Paseo de la Castellana, 179
  28046 MADRID

- GISPERT - PHILIPS - CASIO
  Lagasca, 64
  28001 MADRID

  Provenza, 204-208
  08036 BARCELONA

- IBM
  Paseo de la Castellana, 4
  28046 MADRID

ORGANISMOS Y FUNDACIONES

- ADAMICRO
  Asociación para el Desarrollo de la Tecnología y Aplicaciones de los Microprocesadores
  Sor Angela de la Cruz, 6
  28020 MADRID

- FUNDESCO
  Fundación para el Desarrollo de la Función Social de las Comunicaciones
  Serrano, 187-189
  28002 MADRID

Ejercicio de repaso

Las preguntas siguientes proceden de antiguos exámenes sobre informática de nivel A.

1. *a*) Describa cómo pueden almacenarse en la memoria central tablas de una y dos dimensiones, explicando en cada caso la forma de acceder a un elemento particular de la tabla utilizando:

i) un lenguaje de alto nivel,
ii) un lenguaje ensamblador.

b) Una forma de transmitir un mensaje secreto consiste en dividirlo en pares de letras, ignorando los espacios y signos de puntuación, y representar a continuación cada par por un número de 4 cifras.

Indique cómo almacenaría unas tablas que facilitasen la codificación y dibuje un diagrama de flujo de un programa capaz de codificar un mensaje.

Señale también los cambios necesarios para obtener un programa decodificador.

AEB 79 II

2. *a*) Demuestre, mediante un gráfico o de cualquier otra forma, que la ecuación cúbica

$$x^3 - 3x + 1 = 0$$

tiene tres raíces reales.

Elabore el diagrama de flujo de un programa capaz de obtener las tres raíces por el método de Newton-Raphson.

b) Demuestre que el proceso iterativo

$$x_{k+1} = \frac{3x_k}{2} - \frac{x_k^3}{2a}$$

converge hacia $\sqrt{a}$ y averigüe el orden del proceso.

AEB 79 II

3. *a*) Describa un método para generar una secuencia de números seudoaleatorios distribuidos uniformemente dentro del intervalo [0, 1]. Describa los problemas que pueden presentarse y la forma de resolverlos.

b) En un juego participan dos jugadores y cada uno de ellos lanza por turno al aire un par de monedas.

En cada tirada, dos caras puntúan 2, una cara y una cruz 1 y dos cruces 0. Gana el primero que consiga reunir *exactamente* 20 puntos. Si en una tirada se sobrepasan esos 20 puntos, tal tirada no se cuenta.

i) Determine la distribución de frecuencias de la puntuación para una tirada y, supuesto que cuenta con una subrutina capaz de generar números seudoaleatorios como la descrita en *a*), diseñe el diagrama de flujo de una rutina que produzca dicha puntuación.

ii) Diseñe el diagrama de flujo correspondiente a un programa que simule el juego completo.

AEB 79 II

4. Nombre un lenguaje de programación de alto nivel y escriba en él un programa que satisfaga las siguientes especificaciones: leer una secuencia de enteros no negativos escritos en un formato tal que cada uno pueda ser leído simplemente como un número; no es preciso programar la lectura de cada uno de los caracteres individuales; como finalizador de la secuencia se utiliza un entero negativo. Cada uno de los enteros no negativos corresponde al número de pasajeros que realizan un determinado viaje en tren. El programa tiene que indicar a la salida el número de viajes efectuados con un número de pasajeros comprendido entre 0 y 19, entre 20 y 39, entre 40 y 59, etc., entre 980 y 999 y, por último, con más de 999 pasajeros.

JMB 79 II

5. Cierto proceso de fabricación depende de una temperatura crítica. y se propone construir un sistema informático con un soporte físico especial para controlarlo.

El soporte físico en cuestión mide la temperatura cada 30 segundos y lleva el valor obtenido a una posición X de la memoria central por medio de un transformador analógico-digital.

La acción correctora se inicia por medio de un valor que el ordenador coloca en otra posición Y. Cuando la acción comienza. el propio circuito hace que la posición Y vuelva a 0. Si esto no ocurre. aparece un mensaje de error en el cuadro de mando del ordenador. En cualquier caso. con independencia de que se produzca o no el fallo mencionado. la acción correctora termina en 10 segundos.

Se ha escrito una subrutina para ejecutar la parte central de la función de control. La especificación de dicha subrutina establece que debe leer un valor en la posición X y llevar X a -1. A continuación emprende una acción correctora con arreglo a la siguiente tabla:

Temperatura	*Acción*	*Valor llevado a Y*
menos de 18 °C	acción A	1
entre 18 °C y 21 °C	ninguna	0
más de 21 °C	acción B	2

Demuestre que esta subrutina puede someterse a una prueba completa sin necesidad de recurrir al proceso de manufactura. Indique el conjunto de datos necesario para realizar la prueba exhaustiva propuesta.

UL 78 I

6. Explique qué se entiende por documentación de un sistema y describa lo que aportan a la misma el analista y el programador.

Describa cómo se actualiza la documentación basándose en la modificación de un programa de una serie de varios. Proponga un cambio específico —la necesidad de convalidar un dato de entrada. por ejemplo— y explique qué modificaciones es preciso introducir en la documentación de los programas de la serie.

UL 78 II

7. Elabore un diagrama de flujo y escriba el programa correspondiente en un lenguaje de alto nivel para resolver el siguiente problema:

Una persona percibe por cierto concepto 200.000 pesetas anuales. y cada nuevo año ese ingreso se incrementa en 62.400. El impuesto sobre la renta que debe satisfacer es del 35 por 100. pero quedan exentas las primeras 130.000 pesetas. En el año que estamos considerando. el índice del coste de la vida vale 150. y

cada año se incrementa en un 15 por 100 con respecto al anterior. A partir de esto. el programa debe averiguar el año en que por vez primera el ingreso menos el impuesto dividido por el índice de coste arroje un valor inferior al de este año.

OLE 77 II

8. El ritmo de consumo de agua de un pantano es constante. y los aportes que recibe proceden de la lluvia. La intensidad de las precipitaciones y los intervalos entre ellas tienen una distribución normal. pero el tiempo medio entre borrascas varía según la época del año de acuerdo con una función sinusoidal.

a) Elabore el diagrama de flujo de un programa que simule la cantidad de agua del embalse a lo largo de varios años.

b) Indique cómo habría que modificar el programa para simular el funcionamiento de un aliviadero que entre en acción cuando la cantidad de agua alcance un valor determinado.

c) ¿De qué forma podría utilizarse el programa para determinar el tamaño óptimo del embalse?

OLE 78 II

9. *a*) ¿Qué significa **iteración**?

b) Elabore el diagrama de flujo correspondiente a una rutina de cálculo de raíces cuadradas por iteración. Debe tener en cuenta:

i) las verificaciones a que han de someterse los datos;
ii) la forma de establecer el valor de partida;
iii) el cálculo de las repeticiones sucesivas;
iv) la forma de poner fin al proceso.

Explique por qué el proceso descrito conduce a la solución correcta.

OLE 79 I

10. *a*) Elabore el diagrama de flujo correspondiente a una subrutina capaz de calcular el valor de ${}_a\int^b f(x)\mathrm{d}x$ mediante la aplicación de la regla de los trapecios a una serie de subdivisiones del intervalo comprendido entre a y b.

b) La ecuación diferencial $\mathrm{d}y/\mathrm{d}x = f(x, y)$ puede integrarse utilizando la igualdad

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x, y)\mathrm{d}x$$

para pasar de un punto a otro y aproximándose a la integral por medio de una sola aplicación de la regla de los trapecios.

i) ¿Por qué hay que repetir cada paso para determinar $y(x_{n+1})$ con la mayor exactitud posible?

ii) ¿Cómo podría encontrarse un valor de partida para la iteración?

iii) Elabore un diagrama de flujo del proceso partiendo de la subrutina que tiene escrita o siguiendo cualquier otro procedimiento.

OLE 79 II

11. i) En un río, se hacen lecturas del flujo a intervalos de tres horas. Elabore el diagrama de flujo correspondiente a un programa que calcule por integración la cantidad total de agua que pasa por la estación de medida en una semana.

ii) Elabore el diagrama de flujo correspondiente a un nuevo programa que utilice el anterior como subrutina para calcular la media, la mediana, la moda y la desviación típica de los valores semanales de flujo a lo largo de un año.

OLE 80 I

12. *a*) La posición de un meteoro en el espacio se determina haciendo dos observaciones simultáneas de su marcación desde otras tantas estaciones distintas y midiendo su elevación desde una de ellas. Describa de qué forma pueden transformarse estas observaciones en tres ecuaciones que den las coordenadas x, y, z del meteoro (puede ignorar la curvatura de la tierra).

b) Elabore el diagrama de flujo correspondiente a un programa capaz de resolver esas ecuaciones.

c) Indique cómo podría ampliarse el programa para que determine también la velocidad del meteoro a partir de dos observaciones separadas por un breve intervalo de tiempo.

OLE 80 II

Las instrucciones del Basic

Recogemos en estas páginas los aspectos más importantes de las instrucciones y funciones de la versión de referencia del Basic usada en este libro.

ABS(X)	Calcula el valor absoluto del argumento.
AND	Une dos condiciones de una instrucción **IF.** La condición compuesta sólo es cierta si lo son simultáneamente las dos individuales.
ASC(X$)	Determina el código ASCII del primer carácter del argumento.
ATN(X)	Determina en radianes el ángulo cuya tangente es el argumento.
CHR$(X)	Determina el carácter cuyo código ASCII es igual al argumento.
COS(X)	Determina el coseno del argumento. que se da en radianes.
DATA	Proporciona valores para una instrucción **READ**.
DEF FN	Define una función.
DIM	Declara el tamaño de una o más matrices.
END	Fin del programa.

EXP(X)	Calcula el valor de *e* elevado a la potencia indicada por el argumento.
FOR ... TO	Repite la porción del programa que va hasta la instrucción **NEXT** correspondiente, un número especificado de veces.
GOTO	Transfiere el control al número de línea indicado.
GOSUB	Transfiere el control al subprograma situado en la línea indicada.
IF ... THEN	Si la condición es cierta. transfiere el control a la línea indicada a continuación de **THEN**.
IF ... THEN GOSUB	Si la condición es cierta. transfiere el control al subprograma situado en la línea indicada.
INPUT	Permite introducir datos por el teclado. Incluye una relación de las variables cuyos valores van a introducirse.
INT(X)	Calcula el valor del mayor entero no superior al argumento.
LEN(X$)	Indica el número de caracteres del argumento.
LET	Asigna el valor de la expresión. la variable o la constante situada a la derecha del signo igual, a la variable situada a su izquierda.
LOG(X)	Calcula el logaritmo natural del argumento.
MID$(X$, A, B)	Devuelve una subserie de **X$** con **B** caracteres de longitud que empieza en el carácter de posición **A**.
NEXT	Señala el fin de la porción del programa repetida bajo el control de la instrucción **FOR ... TO.**
NOT	Niega una condición en una instrucción **IF**.
ON ... GOTO	Si la expresión o la variable de control tienen el valor **N**. el control se transfiere a la instrucción correspondiente al número **N** indicado.
ON ... GOSUB	Si la expresión o la variable de control tienen el valor **N**. el control se transfiere al subprograma situado en el número **N** indicado.
OR	Une dos condiciones de una instrucción **IF**. La condición compuesta es cierta con tal que lo sea una cualquiera de las dos individuales.
PRINT	Imprime o presenta en pantalla las variables o constantes indicadas.
READ	Lee los valores de las variables de una instrucción **DATA**.
REM	Advierte que la instrucción a la que precede es una aclaración. Pueden incluirse aclaracio-

	nes en cualquier punto de un programa para mejorar su legibilidad y siempre son ignoradas por el ordenador.
RESTORE	Hace que el ordenador empiece a leer a partir de la primera instrucción **DATA**.
RETURN	Devuelve el control desde un subprograma.
RND(X)	Entrega un número aleatorio uniformemente distribuido en el intervalo 0-1.
SGN(X)	Produce + 1 si **X es positivo** 0 si **X** es cero – 1 si **X** es negativo.
SIN(X)	Calcula el seno del argumento, que se indica en radianes.
SQR(X)	Calcula la raíz cuadrada positiva del argumento.
STEP	Especifica la cantidad que se suma al contador tras cada repetición de un bucle **FOR ... TO**.
STOP	Interrumpe la ejecución del programa.
STR$(X)	Produce una serie de caracteres alfanuméricos equivalente al número que le sirve de argumento.
TAB(X)	Organiza en tablas los datos de salida, que comienzan a imprimirse a **X** espacios del comienzo de la línea.
TAN(X)	Calcula la tangente del argumento, indicado en radianes.
VAL(X$)	Calcula el valor numérico de la serie de caracteres alfanuméricos que le sirve de argumento.

Glosario

Este glosario contiene las definiciones de todos los términos técnicos. de programación y de otras disciplinas. empleados en el texto.

abierto para escritura: dícese de un fichero que está en situación de recibir datos.
abierto para lectura: dícese de un fichero cuyos datos pueden leerse.
actividad: operación identificable que forma parte de una tarea.
alfanumérico, dato: conjunto de caracteres formado por letras y números.
algoritmo: descripción sistemática de los pasos necesarios para ejecutar una tarea.
análisis numérico: rama de las matemáticas que trata de resolver ecuaciones y operaciones como la integración o la diferenciación por métodos aproximados que implican la repetición de cálculos bastante sencillos.
análisis sintáctico: determinación de la estructura de un programa por medio de un compilador o un intérprete.
análisis de sistemas: proceso de determinación de las exigencias planteadas por una operación de tratamiento de datos y de las fases de que consta dicha operación.
análisis de trayectorias críticas: estudio de la red de actividades y

sucesos de una tarea encaminada a determinar la secuencia de los primeros de la que depende críticamente la culminación de la segunda.

árbol: estructura de datos jerárquica en la que cada elemento está unido a otro situado por encima de él y a ninguno o a varios situados por debajo.

árbol binario: aquél cuyos nudos tienen como máximo dos subárboles cada uno.

argumento (de una función): cantidad sobre la que opera esa función.

argumento ficticio: variable empleada en la definición de una función para relacionar un argumento con la fórmula de evaluación de la función.

ASCII (código normalizado estadounidense para el intercambio de información): código de caracteres frecuentemente utilizado en informática.

asignación: operación de adjudicar un valor a una variable.

autodocumentado: dícese del programa cuyas instrucciones son tan claras que cumplen la función de documentación del mismo.

base de una pila: extremo fijo de una pila. opuesto a aquél por el que se efectúa el trasiego de datos.

bifurcación condicional: transferencia del control a un punto determinado del programa cuando se cumplen ciertas condiciones.

bifurcación incondicional: transferencia del control a un punto determinado del programa cualesquiera que sean las condiciones.

bifurcación múltiple: bifurcación condicional a una de entre varias líneas de programa.

bucle: porción de un programa que se ejecuta repetidamente.

bucle anidado: bucle contenido dentro de otro.

bus: véase **canal**.

búsqueda binaria: técnica de búsqueda que consiste en la reducción sistemática de la porción del conjunto de datos que contiene al elemento que se busca.

búsqueda secuencial: técnica de búsqueda que consiste en el recorrido sistemático de todos los elementos de un conjunto de datos hasta localizar el que se busca.

cadena alfanumérica: conjunto de datos. por lo general un subconjunto de un fichero. suficientemente pequeño como para caber en la memoria central del ordenador.

cadena nula: la que no contiene caracteres.

cadena de saltos: secuencia de bifurcaciones condicionales.

campo: unidad de datos de un fichero.

canal: dispositivo de transmisión de datos. direcciones y señales de control en el interior del ordenador.

ciclo de instrucción: secuencia de operaciones necesaria para ejecutar completamente una instrucción de máquina.

cima de una pila: elemento situado en la parte superior de la misma.

cola: estructura de datos en la que se introducen elementos por detrás y se extraen por delante.

coma flotante: representación de un número como producto de una función por una potencia entera de 2 o de 10.

constante: dato que conserva su valor durante el ciclo completo de ejecución del programa.

contador de bucle: variable que controla el número de repeticiones de un bucle dentro de un programa.

dato: información preparada de forma tal que puede introducirse. tratarse. almacenarse o llevarse a la salida en un ordenador.

dato numérico: número.

datos de prueba: conjunto de datos elegidos para poner a prueba el funcionamiento de un programa o de un módulo de programa.

debe: cantidad retirada de una cuenta.

declaración (de una matriz): instrucción por la que se atribuyen un nombre y unas dimensiones a una matriz.

descomposición del tiempo: enfoque general del problema de la creación de modelos matemáticos que consiste en la consideración de todos los cambios que ocurren en dicho modelo durante un intervalo finito de tiempo.

devolver (return): entregar un subprograma el control al programa desde el que se le ha llamado.

documentación: descripción escrita de la forma en que trabaja un programa o de su modo de empleo.

documentación para el operador: descripción de la forma en que debe cargarse un programa en el ordenador.

documentación para el programador: descripción técnica de la estructura y el funcionamiento de un programa.

documentación para el usuario: descripción de la forma en que debe utilizarse un programa.

efecto secundario: alteración inadvertida, como consecuencia de la ejecución de una operación, del valor de una variable no asociada a aquélla.

elemento: cada uno de los datos individuales contenidos en una estructura (una matriz, por ejemplo).

eliminación de Gauss: técnica de resolución de sistemas de ecuaciones.

ensamblador, lenguaje: lenguaje de nivel más alto que el de máquina. pero que todavía actúa directamente sobre los registros y circuitos del ordenador.

entrada: suministro de datos a un ordenador desde el exterior.

error de funcionamiento: el que ocurre durante la ejecución del programa.

error sintáctico: el que se debe al empleo incorrecto del lenguaje de programación.
estado de una cuenta: documento que recoge las entradas y salidas de una cuenta a lo largo de un período de tiempo determinado.
estructura de datos: conjunto de datos asociados mutuamente por ciertas reglas.
evaluar (una función): determinar el valor que toma su argumento.
expresión de control: la utilizada en la bifurcación múltiple.
extender una factura: imprimirla o escribirla.
extraer: sacar el dato que ocupa la parte superior de una pila.

factura: documento que notifica a una persona o entidad la obligación que tiene de pagar por una serie de bienes o servicios recibidos.
fecha primera: la más temprana en la que pueden terminarse todas las actividades que conducen a un hecho.
fecha última: la más tardía en que pueden completarse todas las actividades que conducen a un hecho.
fichero: conjunto organizado de información dotado de una estructura específica.
FIFO *(First In, First Out)*, **primero-en-entrar—primero-en-salir:** véase **cola**.
función: transformación de una cantidad (el argumento) en otra (el valor de la función).

haber: cantidad abonada a una cuenta.
hecho: etapa cubierta en la realización de una empresa o una tarea.

implementación: versión del Basic aceptada por un tipo concreto de ordenador.
indicación: mensaje que recuerda al usuario lo que debe hacer.
indicador de fin de datos: dato que señala el final de un conjunto de ellos.
indicador de fin de fichero: conjunto único de caracteres que señala el final de un fichero.
índice (de una matriz): número que identifica un elemento particular de la misma.
instrucción: operación que debe ejecutar un programa escrita por el programador en un lenguaje adecuado.
integración numérica: obtención del valor aproximado de una integral por medio de una secuencia de cálculos relativamente sencillos.
interactivo: dícese de un programa o un lenguaje de programación que facilita la "conversación" entre el usuario y el ordenador.
interfaz: punto de contacto entre un elemento del ordenador (físico o lógico) y otro o entre un elemento y el exterior.

interfaz con el usuario: punto o puntos de contacto entre el usuario y un programa o un ordenador.
interrupción: instrucción de un programa que ocasiona la detención del mismo en un punto determinado.
introducir: añadir un nuevo elemento a una pila.
inversa, función: la que realiza la transformación opuesta a la provocada por aquélla de la que es inversa.
iteración: cada uno de los pasos del ciclo de cálculos repetitivos que convergen hacia un resultado con un margen de precisión dado.

lenguaje de alto nivel: lenguaje de aplicación en el que resulta cómodo y sencillo describir las estructuras de información y la secuencia de operaciones necesarias para realizar una tarea determinada.
lenguaje máquina: el que controla directamente el soporte físico del ordenador.
LIFO *(Last In, First Out)*, **último-en-entrar—primero-en-salir:** véase **pila.**
lista: estructura ordenada de datos en la que éstos pueden introducirse y extraerse en cualquier posición.

llamada: transferencia del control a un subprograma.

mantenimiento del programa: correcciones que se hacen a un programa para mejorarlo o para ponerlo al día.
margen: diferencia entre las fechas primera y última para un hecho particular.
matriz: conjunto de datos homogéneos almacenados de manera que constituyen una unidad.
matriz bidimensional: la que tiene dos juegos de índices; sus elementos pueden considerarse ordenados en filas y columnas.
memoria circular: área fija de la memoria dentro de la que la inserción y la extracción de elementos tienen lugar de manera que la cabeza y la cola van girando al unísono.
modelo: simplificación deliberada y ordenada de un sistema que sirve para aislar sus rasgos de comportamiento esenciales.
modelo de compartimientos: modelo muy usado para simular ecosistemas.
modelo matemático: el expresado en términos matemáticos.
módulo: parte de un programa que realiza una tarea específica.

Newton, método de: técnica iterativa de resolución de ecuaciones.
nudo: elemento de un árbol.
número de línea: entero que se coloca al principio de cada una de las líneas de un programa para identificarlas.

ordenación por el método burbuja: técnica de clasificación que consiste

en hacer que los números menores asciendan por "flotación" hasta la parte delantera del conjunto de datos.

ordenación por selección: técnica de clasificación que consiste en escoger el número mayor (o el menor) de un subconjunto de datos cada vez menor.

palabra de instrucción: palabra usada para identificar un tipo particular de instrucción en Basic.

parámetro: variable utilizada para transferir información a o desde un subprograma.

pase de prueba: ejecución a mano de todas las instrucciones de un programa.

paso: cantidad en que se incrementa el contador de un bucle tras cada repetición de éste.

pila: estructura de datos en la que sólo pueden introducirse o extraerse elementos por un extremo.

procedimiento: véase **subprograma**.

proceso: manipulación de los datos por el ordenador, que los somete a operaciones como la clasificación, la selección o la ejecución de cálculos.

programa: conjunto de instrucciones que se da a un ordenador.

programa principal: parte de un programa desde la que se llama a los subprogramas.

puntero: dato que sirve para indicar la posición de otro.

puntero nulo: el que no señala nada.

raíz: nudo superior de un árbol.

recibo: documento que se entrega contra la recepción de una cantidad de dinero.

recorrido de un árbol: acto de "visitar" cada uno de sus nudos de forma sistemática.

recurrencia: capacidad que tienen un subprograma o una función de llamarse a sí mismos.

red: diagrama que recoge las interrelaciones entre las actividades y los hechos que conducen a la culminación de una empresa.

refinamiento por etapas: técnica de diseño de programas que consiste en descomponer la tarea en pasos cada vez más breves hasta alcanzar el grado de detalle suficiente para escribir el programa.

refundición-ordenación: ordenación por medio de refundiciones sucesivas.

refundir: combinar dos conjuntos de datos ordenados de manera que resulte uno solo también ordenado.

registro: en un fichero, unidad correspondiente a un dato; también, espacio de almacenamiento reservado a un dato concreto que sirve para un propósito determinado.

resolución (gráfica): tamaño del menor espacio de la pantalla controlable independientemente por un programa de realización de gráficos.

saldo: cantidad neta que hay en una cuenta.

salida: entrega de datos que hace un ordenador al medio externo.

semilla: número utilizado para lanzar el generador de números aleatorios.

Simpson, regla de: técnica aproximativa de integración que utiliza una serie de curvas parabólicas.

sistema de ecuaciones: conjunto de varias ecuaciones con varias incógnitas que tienen una (o muchas) solución(es) común(es).

sistema operativo: programa o conjunto de programas que organizan el soporte físico de un ordenador con arreglo a ciertos objetivos.

soporte lógico: conjunto de instrucciones que se dan a un ordenador.

subárbol: subconjunto de un árbol.

subprograma: parte de un programa que ejecuta una tarea específica y a la que se transfiere el control desde cualquier punto del programa principal; realizada la tarea, devuelve el control al punto desde el que fue llamado.

tiempo real, programa en: el que acomoda su ritmo al de algún hecho externo.

trapecios, regla de los: técnica numérica de integración que se aproxima a la función que ha de integrarse por medio de una serie de rectas.

trasposición, método de: técnica iterativa de resolución de ecuaciones.

validación: proceso de verificación de un dato de entrada antes de su almacenamiento o su procesado.

valor (de una función): cantidad producida por la función cuando se aplica a un argumento (o a varios).

variable: dato cuyo valor puede cambiar a lo largo del programa.

variable de control: la que se usa en una bifurcación múltiple.

variable global: la que tiene un intervalo de validez que cubre todo el programa en el que aparece.

variable local: la que tiene un intervalo de validez limitado a un subprograma.

versión de referencia: versión del Basic utilizada en este libro; es independiente de cualquier marca de ordenador y contiene todos los elementos y características esenciales del lenguaje.

Observaciones para el profesor

Estas observaciones constituyen una guía para profesores de programación que imparten clases a nivel A o equivalente, por lo general como parte de un curso más amplio sobre informática. Dan algunas ideas sobre la mejor forma de utilizar el libro y contienen las soluciones de algunas de las preguntas de los ejercicios propuestos al final de cada uno de los capítulos. En cualquier caso, lo más importante para sacar partido al texto es familiarizarse con él que, en parte, se ha pensado como una especie de "guión" para el profesor.

Cómo usar este libro

Dado que el texto es bastante más que una simple introducción a los aspectos característicos del Basic, puede utilizarse con cierta flexibilidad.

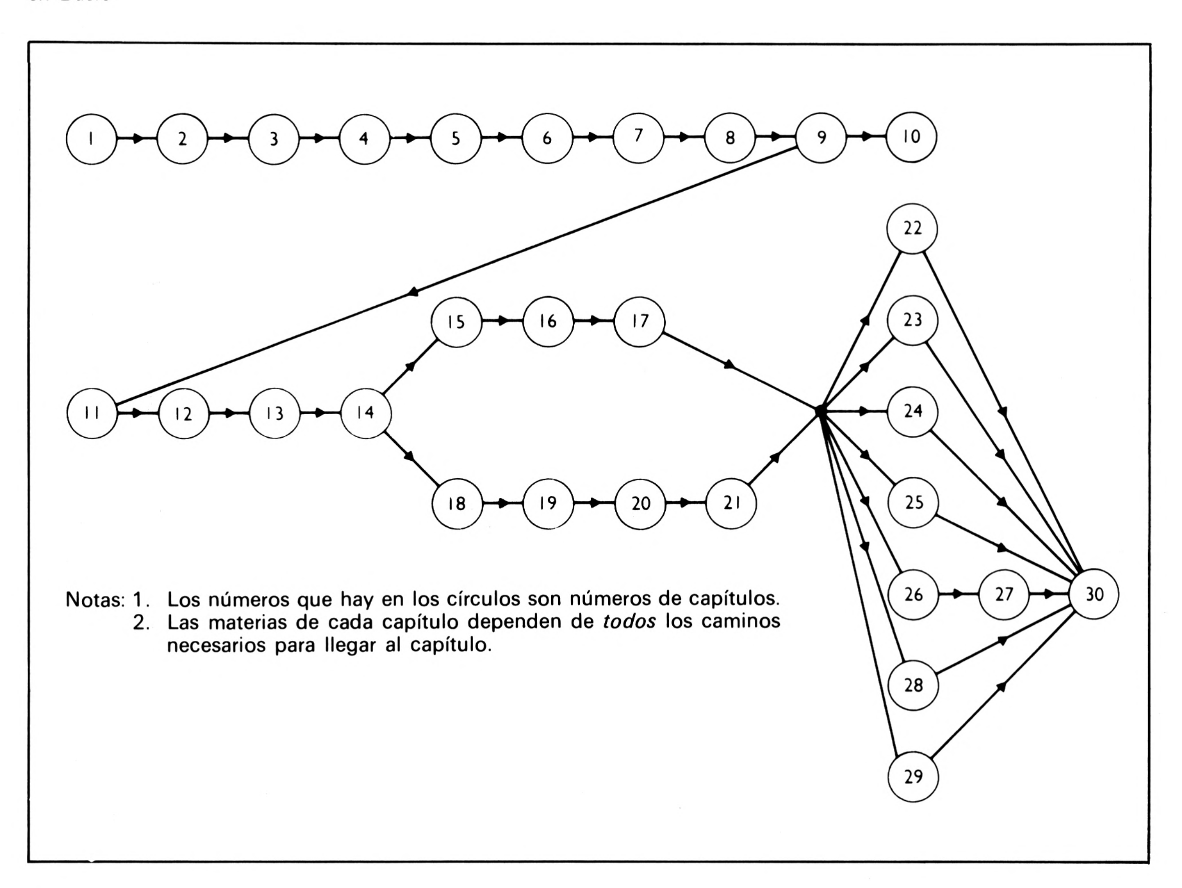
Observe la figura 32.1: los diez primeros capítulos, que presentan los elementos del lenguaje y algunas técnicas rudimentarias de programación, deben estudiarse en secuencia. El capítulo 10, dedicado a la manipulación de ficheros, es opcional, puesto que este aspecto depende del ordenador con que se trabaje.

Los cuatro capítulos referentes al diseño de programas (11 a 14) constituyen, por muchas razones, el núcleo del texto, y deben estudiarse seguidos y muy a fondo; es imprescindible dominarlos completamente para poder seguir adelante. El orden de los siete siguientes puede alterarse: o bien se estudian primero las operaciones fundamentales de programación (15 a 17) o bien se empieza por las estructuras de datos (18 a 21).

Los capítulos de aplicaciones de la programación (22 a 29) son autónomos, salvo los 26 y 27, que están relacionados entre sí. Pueden, por tanto, estudiarse en cualquier orden y no hay inconveniente en prescindir de los que no se consideren interesantes. El último capítulo está formado por una serie de propuestas de trabajos, y guarda con los anteriores una relación bastante laxa.

Si se desea dar al curso una carácter menos teórico, pueden omitirse o estudiarse superficialmente los capítulos 15 a 21, dedicados a operaciones de programación y estructuras de datos; si se procede así, se resentirán algunos otros capítulos, pero en general, casi todos podrán estudiarse sin problemas.

Figura 32.1
Dependencia entre capítulos en *Programación avanzada en Basic*



Respuestas a los ejercicios

La selección de preguntas respondidas se ha hecho con arreglo a los siguientes criterios:

- Se han incluido las respuestas que pueden exponerse de forma breve y específica, con la excepción de las correspondientes a definiciones de términos, que pueden encontrarse en el glosario.
- De las preguntas que exigen cierta discusión, se han incluido los aspectos más relevantes, pero en la mayor parte de los casos no se han extraído conclusiones, porque lo importante de una respuesta es su lógica, no lo acertado de la solución final.
- Sólo se incluyen los listados correspondientes a programas bastante cortos y los que respondan a técnicas específicas.
- No se incluyen las respuestas correspondientes a preguntas planteadas en exámenes de informática.
- De las preguntas que implican la comprensión de una parte del texto, se da como respuesta la referencia a esa parte.

Ejercicio 1

2. *a*) Véanse las secciones 1.3, 1.4 y 1.5.

3. En ambos casos se realiza una tarea específica con una máquina de tipo general.

Ejercicio 2

2. Nombres incorrectos: **A$2, 3A, $M, AB, A14.**

3. 6 es una constante numérica que puede utilizarse en cálculos. "6" es una constante alfanumérica que puede utilizarse en operaciones con esta clase de caracteres.

4. Identificar el programa, el autor, la versión y los datos.
Identificar una parte de un programa.
Explicar el funcionamiento de una instrucción.
Establecer el empleo de una o más variables.
Crear espacio para la ulterior inserción de nuevas instrucciones.

5. Véase la sección 2.1.

Ejercicio 3

2. *a*) **C** = 5
b) **D** = 6

3. Instrucciones correctas:

```
10 PRINT "QUE HORA ES?"
20 INPUT H$
30 PRINT "SON LAS"; H$
```

4. **A** = 4, **B** = 4
B = 3, **A** = 3

5. *a*) Para intercambiar los valores de las variables **A** y **B** se utiliza la variable provisional **T**:

```
10 LET T = A
20 LET A = B
30 LET B = T
```

b) Volviendo a usar la variable provisional **T**:

```
10 LET T = A
20 LET A = C
30 LET C = B
40 LET B = T
```

6. **C** = 2E3

7. *b*) `(A - C) * (B + 9)` = (10 − 8) * (5 + 9)
= 2 * 14
= 28

f) `C/(A - 2 * B)` = 8/(10 × 2 * 5)
= 8/(10 − 10)
= 8/0 imposible dividir por cero.

8. Véase el programa ejemplo 3.2.

9.

```
100 REM EJERCICIO 3, PREGUNTA 9
105 REM CALCULO DE IVA
110 REM
200 PRINT "CALCULO DE IVA"
205 PRINT
210 PRINT "CANTIDAD DE DINERO?";
215 INPUT C
220 PRINT "TIPO DE IVA?";
225 INPUT T
230 LET V = C * T/100
235 LET A = C + V
240 PRINT "IVA:"; V
245 PRINT "CANTIDAD + IVA:"; A
250 END
```

10.

```
100 REM EJERCICIO 3, PREGUNTA 10
105 REM PROGRAMA EJEMPLO 3.3
```

```
110 REM CON ACUMULACION DE VARIAS SENTENCIAS EN
        UNA LINEA
115 REM
200 PRINT "CALCULOS DE INTERES COMPUESTO": PRINT
    :PRINT
205 PRINT "CANTIDAD AHORRADA (PESETAS)": INPUT P
210 PRINT "INTERES": INPUT R
215 PRINT "TIEMPO DE IMPOSICION": INPUT T: PRINT
220 LET A = P*(1 + R/100)^T
225 PRINT "DINERO ACUMULADO :";A;" PESETAS"
    : PRINT
230 END
```

11.

```
100 REM EJERCICIO 3, PREGUNTA 11
105 REM CALCULOS DE UN DEPOSITO CILINDRICO
110 REM
200 PRINT "CALCULOS DE UN DEPOSITO CILINDRICO"
205 PRINT
210 PRINT "RADIO INTERIOR?";
215 INPUT R
220 PRINT "GROSOR?";
225 INPUT G
230 PRINT "ALTURA INTERIOR?";
235 INPUT H
300 REM CALCULOS
305 REM CALCULO DE LA CAPACIDAD
310 LET V1 = 3.14159 * R * R * H
315 REM CALCULO DEL VOLUMEN EXTERIOR
320 LET V2 = 3.14159 * (R + G) * (R + G) * (H +
        G)
325 REM CALCULO DEL VOLUMEN DE HORMIGON
330 LET V3 = V2 - V1
400 REM SALIDA
405 PRINT "CAPACIDAD DEL DEPOSITO:"; V1
410 PRINT "VOLUMEN DE HORMIGON:"; V3
415 END
```

Ejercicio 4

1.

```
100 IF A > B THEN 115
105 LET X = B
110 GOTO 120
115 LET X = A
120 REM CONTINUAR
```

o bien

```
100 IF A > B THEN X = A ELSE X = B
```

2. *b)* `(X >= 5) AND (X <= 20)`

   c) `(X > 0) AND (X < 45)`

d) `(X$ >= "A") AND (X$ <= "Z")`

e) `((X >= 48) AND (X <= 57) OR (X >= 65) AND (X <= 70))`

f) `(X = INT(X)) AND ((X >= 1) AND (X <= 5) OR (X > 10))`

g) `((X + Y) > 100) OR ((X + Y) < 50)`

3. *a*) **A** = 2

b) No.

c)
```
500 IF ((X >= 5) AND (X <= 10)) OR ((Y < 5) OR
        (Y >10)) THEN 540
```

d) Nada.

e)
```
500 IF ((X >= 5) AND (X <= 10)) OR ((Y < 5) OR
        (Y >10)) THEN A = 2 ELSE A =1
```

4. Véase el programa ejemplo 4.1.

5. *b*)
```
10 IF (C$ >= "A") AND (C$ <= "Z") AND (LEN(C$)
        = 1) THEN 30
20 LET C$ = "?"
30 REM CONTINUAR
```

c)
```
10 IF X >= .000001 THEN 30
20 LET X = 0
30 REM CONTINUAR
```

d)
```
10 IF (A + B) <= 5000000 THEN 40
20 LET A = 2500000
30 LET B = 2500000
40 REM CONTINUAR
```

e)
```
10 IF A$ <> B$ THEN 40
20 LET A$ = "E"
30 LET B$ = "E"
40 REM CONTINUAR
```

f)
```
10 IF X <= 10 THEN 30
20 LET X = 10
30 IF X >= 1 THEN 50
40 LET X = 1
50 REM CONTINUAR
```

6. *a*)
```
100 REM EJERCICIO 4, PREGUNTA 6
105 REM SOLUCION DE UNA ECUACION CUADRATICA
110 REM
200 REM SEGMENTO DE ENTRADA
205 PRINT "SOLUCION DE LA ECUACION CUADRATICA"
210 PRINT "INTRODUZCA LOS VALORES DE A, B Y C"
215 INPUT A, B, C
300 REM COMPROBAR SI LA SOLUCION EXISTE
305 IF B * B >= 4 * A * C THEN 400
310 PRINT "NO EXISTE SOLUCION"
315 STOP
400 REM CALCULO
405 LET D = SQR(B * B - 4 * A * C)
410 LET X1 = (-B + D)/(2 * A)
415 LET X2 = (-B - D)/(2 * A)
```

```
500 REM SALIDA
510 PRINT "SOLUCIONES:"
515 PRINT "X1 = "; X1
520 PRINT "X2 = "; X2
525 END
```

8.

```
100 REM EJERCICIO 4, PREGUNTA 8
105 REM SENCILLA CALCULADORA
110 REM
200 REM ENTRADA
205 PRINT "PRIMER NUMERO?";
210 INPUT N1
215 PRINT "OPERACION?";
220 INPUT P$
225 PRINT "SEGUNDO NUMERO?";
230 INPUT N2
300 REM BIFURCACION A UNA OPERACION
305 IF P$ = "+" THEN 405
310 IF P$ = "-" THEN 415
315 IF P$ = "*" THEN 425
320 IF P$ = "/" THEN 435
325 PRINT "OPERACION NO VALIDA"
330 PRINT "POR FAVOR COMIENCE DE NUEVO"
335 GOTO 200
400 REM CALCULOS
405 LET R = N1 + N2
410 GOTO 500
415 LET R = N1 - N2
420 GOTO 500
425 LET R = N1 * N2
430 GOTO 500
435 LET R = N1 / N2
500 REM SALIDA
505 PRINT "RESULTADO"; R
510 PRINT
515 PRINT "DESEA REALIZAR OTRO CALCULO?";
520 INPUT R$
525 IF R$ = "SI" THEN 200
530 END
```

Ejercicio 5

2. *b*) 20 veces.
c) **N** veces.
d) 19 veces.
e) 7 veces.
f) 5 veces.
g) **INT(Y − X)/(Z + 1)** veces.

3.

```
10 FOR N = 1 TO 10
20 INPUT N$, T, R
30 NEXT N
```

4. La línea 130 debe quedar

```
130 LET C = N
```

5. *a*) Hay que intercambiar las variables de las líneas 520 y 525.
   b) 40 veces.

7.
```
100 REM EJERCICIO 5, PREGUNTA 7
105 REM DETERMINAR EL MENOR ENTERO POSITIVO
110 REM CUYO CUADRADO ES SUPERIOR A 100.000
115 LET N = 100
120 LET N = N + 1
125 IF N * N < 100000 THEN 120
130 PRINT N
135 END
```

8. La nueva versión tiene un bucle del tipo "repetir hasta que" en lugar del tipo "repetir mientras". Desde el punto de vista del usuario. el conjunto final de datos debe contener un nombre, una dirección y un número de teléfono en lugar de sólo un nombre.

9. Insertar la línea 142:

```
142 LET X = 1
```

Sustituir la 150 por:

```
150 LET X = X * (1 + R/100)
```

Ejercicio 6

2.
```
LEN(C$) = LEN(A$) + LEN(B$)
```
3.
```
10 LET D$ = CHR$ (ASC(C$) + 1)
```
4.
```
100 FOR N = 65 TO 90
110 PRINT CHR$(N);
115 NEXT N
```
6.
```
6500 REM VALIDACION DE FECHA D1$
6505 IF LEN (D1$) <> 8 THEN 6600
6510 IF MID$ (D1$,3,1) <> "/" THEN 6600
6515 IF MID$ (D1$,6,1) <> "/" THEN 6600
6520 LET C = 0
6525 FOR I = 1 TO 7 STEP 3
6530 FOR J = I TO I + 1
6535 LET K = ASC (MID$ (D1$,J,1))
6537 REM PRUEBA PARA DIGITO DECIMAL
6540 IF K >= 48 AND K <= 57 THEN 6560
6545 LET C = 1
6550 LET J = I + 1
6555 LET I = 7
```

```
6560 NEXT J
6565 NEXT I
6570 IF C = 1 THEN 6600
6580 IF VAL (MID$ (D1$,1,2)) > 31 THEN 6600
6585 IF VAL (MID$ (D1$,4,2)) > 12 THEN 6600
6590 LET V = 1: REM FECHA CORRECTA
6595 GOTO 6605
6600 LET V = 0: REM FECHA INCORRECTA
6605 REM CONTINUAR
```

Observe que en todos los casos se comprueba si la fecha es válida. Todas las condiciones de esas pruebas remiten el control a la línea 6600.

Ejercicio 7

2. Véase la sección 7.1.

6.

```
5000 REM MULTIPLICACION DE LAS MATRICES A Y B
5005 IF C1 = F2 THEN 5025
5010 PRINT "LAS MATRICES NO SON COMPATIBLES"
5015 PRINT "PARA LA MULTIPLICACION"
5020 GOTO 2000
5025 FOR I = 1 TO F1
5030 FOR J = 1 TO C2
5035 LET C(I, J) = 0
5040 FOR K = 1 TO F2
5045 LET C(I, J) = C (I, J) + A (I, K) * B (K, J)
5050 NEXT K
5055 NEXT J
5060 NEXT I
5065 GOTO 2000
```

Ejercicio 8

2. No. Véanse las secciones 8.3 a 8.16.

3. *a*) `10 LET X = ABS(B - A)`

b) `10 LET X = INT(10 * Y + 0.5)/10`

c) `10 LET M = INT(RND(1) * 100) + 1`

d) `10 LET X = ABS(SGN(Y))`

e) `10 LET H = SQR(A * A + B * B)`

f) `10 LET D = 180 * ATN(T)/3.1415926`

8.

```
100 REM EJERCICIO 8, PREGUNTA 8
105 REM SIMULACION DE UNA CENTRAL TELEFONICA
110 REM INICIALIZACION DEL GENERADOR DE NUMEROS
        ALEATORIOS
```

```
200 PRINT "MINUTOS EN QUE SE RECIBEN LAS LLAMA
          DAS"
205 LET T = 0
210 FOR M = 1 TO 60
215 IF RND(1) > 0.3 THEN 230
220 LET T = T + 1
225 PRINT M
230 NEXT M
300 PRINT
305 PRINT "TOTAL DE LLAMADAS"; T
310 PRINT "MEDIA DE LLAMADAS POR MINUTO"; T/60
315 END
```

9.
```
10 DEF FNL$(U$) = CHR$(ASC(U$) + 32)
20 DEF FNU$(L$) = CHR$(ASC(L$) - 32)
```

10.
```
100 FOR K = A TO B STEP SGN(B - A) * ABS(C)
```

Ejercicio 9

2. No hay instrucciones para señalar el comienzo del subprograma:

   - Los subprogramas no pueden identificarse mediante un nombre.
   - No pueden transferirse parámetros.
   - No es posible escribir subprogramas recursivos.
   - No hay variables locales.

4.
```
100 REM EJERCICIO 9, PREGUNTA 4
105 REM PROBABILIDADES DE CANCELACION DE UN TREN
110 REM
200 REM ENTRADA/CONVALIDACION
205 PRINT "PROBABILIDAD DE CANCELACION?";
210 INPUT P
215 IF P >= 0 AND P <= 1 THEN 300
220 PRINT "INCORRECTO, POR FAVOR VUELVA A INTRO
          DUCIRLO"
225 GOTO 205
300 REM CALCULO/SALIDA
305 PRINT "NUMERO CANCELADO"; TAB(20); "PROBABI
          LIDAD"
310 REM CALCULO DEL FACTORIAL DE 5
315 LET N = 5
320 GOSUB 1000
325 LET F1 = F
330 FOR R = 0 TO 5
335 REM CALCULO DEL FACTORIAL DE (5 - R)
340 LET N = 5 - R
345 GOSUB 1000
350 LET F2 = F
```

```
355 REM CALCULO DEL FACTORIAL DE R
360 LET N = R
365 GOSUB 1000
370 LET F3 = F
375 REM CALCULO DE PROBABILIDAD
380 LET C = F1/F2/F3 * P ^ R * (1 - P) ^ (5 - R)
385 PRINT R; TAB(20); C
390 NEXT R
395 STOP
1000 REM SUBPROGRAMA FACTORIAL
1005 REM EDITADO AQUI
1500 END
```

7. **Subprograma: observaciones sobre el diseño**
El subprograma acepta una serie de caracteres alfanuméricos **L$** y devuelve otra **U$**. Los caracteres se transfieren de la cadena alfanumérica de entrada a la de salida uno por uno, y en la operación se sustituyen todas las minúsculas por mayúsculas. Los demás caracteres no sufren ninguna alteración.

```
1000 REM EJERCICIO 9, PREGUNTA 7: SUBPROGRAMA
1005 REM PARAMETROS
1010 REM L$: ENTRADA DE CADENA ALFANUMERICA
1015 REM U$: SALIDA DE CADENA ALFANUMERICA
1020 REM
1100 REM VERIFICAR LA ENTRADA DE LA CADENA ALFANU
         MERICA, CARACTER POR CARACTER
1105 LET U$ = ""
1110 FOR K = 1 TO LEN(L$)
1115 LET T$ = MID$(L$, K, 1)
1120 REM VERIFICACION PARA LAS MINUSCULAS
1125 LET T = ASC(T$)
1130 IF T < 97 OR T > 122 THEN 1145
1135 REM CAMBIO A MAYUSCULAS
1140 LET T$ = CHR$(T - 32)
1145 LET U$ = U$ + T$
1150 NEXT K
1155 RETURN
```

Ejercicio 10

3.
```
100 REM EJERCICIO 10, PREGUNTA 3
105 REM ACTUALIZACION DE UN FICHERO DE NOMBRES,
        DIRECCIONES Y NUMEROS DE TELEFONO
110 REM
115 DIM U$(20)
120 PRINT "ACTUALIZACION DE UN FICHERO DE NOM
           BRES DIRECCIONES Y NUMEROS DE TELE
           FONO"
125 PRINT
```

```
200 REM ENTRADA DE NOMBRES DE REGISTROS PARA SER
        ACTUALIZADOS
205 PRINT "ESCRIBA LOS NOMBRES DE LOS REGISTROS
          QUE VAN A SER ACTUALIZADOS"
210 PRINT "(UN MAXIMO DE 20)"
215 PRINT "ESCRIBA 'FINARCHIVO' PARA FINALIZAR"
220 FOR K = 1 TO 20
225 INPUT I$
230 IF I$= "FINARCHIVO" THEN 242
235 LET U$(X) = I$
240 GOTO 250
242 LET J = K - 1
245 LET K = 20
250 NEXT K
300 REM COPIA DE REGISTROS, QUE NO VAN A SER
        ACTUALIZADOS, EN UN FICHERO TEMPORAL
305 OPEN "TEMPGUIA" FOR APPEND AS # 11
310 OPEN "NOMDITELGUIA" FOR INPUT AS #10
315 INPUT #10, N$, D$, T$
320 IF N$ = "FINARCHIVO" THEN 400
325 REM BUSQUEDA DEL NOMBRE EN LA MATRIZ
330 LET F = 0
335 FOR K = 1 TO J
340 IF N$ <> U$(K) THEN 355
345 LET F = 1
350 LET K = J
355 NEXT K
360 IF F = 1 THEN 315
365 WRITE #11, N$, D$, T$
370 GOTO 315
400 REM ENTRADA Y ARCHIVACION DE NUEVOS REGIS
        TROS EN EL FICHERO
405 PRINT "ESCRIBA LOS NUEVOS NOMBRES, DIRECCIO
          NES Y TELEFONOS"
410 PRINT "UNO CADA VEZ"
415 PRINT "FINALICE LA ENTRADA CON 'FINARCHIVO',
          'FINDIRECCION', 0000"
420 INPUT N$, D$, T$
425 WRITE #11, N$, A$, T$
430 IF N$ <> "FINARCHIVO" THEN 420
435 CLOSE #11
500 REM CAMBIO DE NOMBRE DEL ARCHIVO TEMPORAL
505 NAME "NOMDITELGUIA" AS "TEMPGUIA"
510 END
```

Ejercicio 11

1. Véase la sección 11.2.
2. Véase en la sección 11.2 el párrafo titulado “Estructura clara”.
3. Ir de “arriba hacia abajo” significa pasar de lo general a lo detallado y de las fases generales de una tarea a los pasos concretos en que se subdivide ésta.

Ir de "abajo hacia arriba" es decidir primero de qué forma se realizarán una serie de operaciones específicas y combinarlas a continuación.

4. *b*) La descomposición alternativa más obvia es:

Introducir los elementos de la primera matriz.
Introducir los elementos de la segunda matriz.
Calcular y mostrar la matriz suma.

c) Los programas difieren en que en la segunda versión las partes de salida y cálculo están combinadas en un único módulo. De esta forma el programa resulta más corto, pero también más difícil de leer.

5. Una descomposición en pasos sería la siguiente:

Crear una matriz para los caracteres de la serie y otra para el número de veces que aparece cada uno.
Para cada uno de los caracteres, repetir el proceso.

Si el carácter no está ya en su matriz correspondiente, colocarlo en ella;

contar el número de veces que aparece en el resto de la cadena alfanumérica;
llevar el resultado a la matriz de números.

Mostrar los contenidos de ambas matrices.

Ejercicio 12

3. *a*) Pase de prueba con los valores **B** = 6, **A** = 3.

	B	**A**	**Q**	**B**	**A**	**Q**
`1010 LET Q = 0`	6	3	0			
`1015 LET B = B - A`	3	3	0	0	3	1
`1020 LET Q = Q + 1`	3	3	1	0	3	2
`1025 IF B > Q THEN 1015`						
`1030 LET B = B + A`				3	3	2
`1035 LET Q = Q - 1`				3	3	1

El resultado —cociente = 1, resto = 3— es incorrecto.

b) El error lógico está en la línea 1025, que debe leerse

```
1025 IF B >= 0 THEN 1015
```

4. Por lo general, es mejor trabajar de los módulos de servicio hacia arriba para determinar las condiciones bajo las que funcionará el programa.

Módulo de servicio **CARGAR**: funciona para cualquier valor de **L$**, pero sólo para valores de **I** comprendidos entre 1 y 30.

Módulo de servicio **ACCEDER**: sólo funciona para valores de **J** comprendidos entre 1 y 30.

Módulo de servicio **BUSCAR**: funciona para todos los valores de **N$** y devuelve valores de **K** comprendidos entre 0 y 30.

Módulo de inicialización: si no recibe ningún parámetro, llama al módulo de servicio **CARGAR** con valores comprendidos entre 1 y 30.

Módulo de mando **INTRODUCIR**: acepta cualquier valor de **D$** y lo pasa al módulo **CARGAR**. Devuelve siempre 1 como valor de **C**. Recibe el valor de **K** del módulo **BUSCAR** y, si no es cero, lo pasa al módulo **CARGAR**.

Módulo de mando **SACAR**: acepta cualquier valor de **D$** y lo pasa al módulo **BUSCAR**. Devuelve siempre 1 como valor de **C**. Recibe el valor de **K** del módulo **BUSCAR** y, si no es cero, lo pasa al módulo **CARGAR**.

Módulo de mando **INDAGAR**: acepta cualquier valor de **D$** y lo pasa al módulo **BUSCAR**. Devuelve siempre 1 como valor de **C**. Recibe el valor de **K** del módulo **BUSCAR**.

Módulo de mando **VISUALIZAR**: no recibe ningún parámetro, entrega siempre 1 como valor de **C**. Pasa al módulo **ACCEDER** un valor de **J** comprendido entre 1 y 30.

Módulo **DETENER**: no recibe ningún parámetro.

Módulo de control: recibe cualquier cadena de caracteres alfanuméricos **C$**, iguala **D$** a los 6 últimos caracteres de **C$** o a la cadena completa si tiene menos de 8 caracteres de longitud. Llama a un módulo de mando si el nombre de la orden es igual a los primeros caracteres de **C$**. Utiliza el valor 1 de la variable **C** para asegurarse de que se ha llamado al módulo de mando.

Conclusiones generales: como puede observarse, los módulos de nivel más alto satisfacen todas las condiciones de funcionamiento correcto de los módulos inferiores. El único inconveniente —que de todas formas no provocaría el fallo del programa— es que puede introducirse y almacenarse cualquier identificación incorrecta.

2. Véanse las secciones 13.2, 13.3, 13.7 y 13.8.
4. Programa ejemplo 11.2: módulo de servicio **BUSCAR**.

Función

Busca en una matriz un elemento igual a la variable **N$**. Entrega el índice del elemento, si lo encuentra, en forma de variable **K**, o un valor cero para esta variable si no lo encuentra.

Estructura

Se utiliza un bucle **FOR ... TO** para explorar la formación. El bucle termina en cuanto se encuentra el elemento buscado.

Método de funcionamiento

Se llama a un subprograma al que se asigna el parámetro **N$**. El algoritmo de esta operación es:

Suponer el fallo de la operación asignando a la variable **K** el valor 0.
Para cada elemento de la matriz, repetir el proceso.
Si el elemento es igual a **N$**,
dar a **K** el valor de su índice;
igualar el valor del índice al límite de la matriz.

El subprograma hace la variable **K** igual a cero si no encuentra el elemento en cuestión, y la iguala al valor de su índice si lo encuentra.

Ejercicio 14

2. *a*) La línea 105 se convierte en:

```
105 FOR K = 1 TO 20
```

y la 125 en:

```
125 LET A = T/20
```

b) La línea 105 es sustituida por:

```
102 PRINT "NUMERO DE NUMEROS?"
103 INPUT M
105 FOR K = 1 TO M
```

y la 125 por:

```
125 LET A = T/M
```

c) Si se usa 0 como indicador de fin de datos, el programa adoptará la forma:

```
100 LET T = 0
105 LET M = 0
110 INPUT N
115 IF N = 0 THEN 135
120 LET T = T + N
125 LET M = M + 1
130 GOTO 110
135 LET A = T/M
```

4. Las líneas que siguen a la 1025 se transforman como sigue:

```
1025 LET C = 0
1030 IF O$ = "RECIBIR" THEN LET C = 1
1035 IF O$ = "ENTREGAR" THEN LET C = 2
1040 IF O$ = "SOLICITAR" THEN LET C = 3
1045 IF O$ = "REALIZAR" THEN LET C = 4
1050 IF C <> 0 THEN 1065
1055 PRINT "ORDEN NO RECONOCIDA. POR FAVOR VUELVA
          A INTRODUCIRLA";
1060 GOTO 1020
1065 ON C GOSUB 2000, 3000, 4000, 5000
1070 GOTO 1020
```

Ejercicio 15

1. *a*)

```
405 LET J = 0
475 LET J = J + 1
655 PRINT "NUMERO DE INTERCAMBIOS"; J
```

2. *a*)

```
402 LET P = 0
420 FOR K = 1 TO M - P - 1
485 LET P = P + 1
```

7. *a*)

```
100 REM EJERCICIO 15, PREGUNTA 7
105 REM CREACION DE UNA TABLA DE BARAJADO
110 REM
115 DIM H$(25)
200 REM INTRODUCCION DE LA LONGITUD DE LA TABLA
        BORRADO DE ELEMENTOS
```

```
205 PRINT "CUANTOS ELEMENTOS?";
210 INPUT N
215 FOR K = 1 TO N
220 LET H$(K) = "**"
225 NEXT K
300 REM INTRODUCCION DE LOS ELEMENTOS Y ARCHIVA
        DO DE LOS MISMOS
305 INPUT X$
310 LET P = INT((ASC(X$) - 64) * N/26)
315 IF H$(P) = "**" THEN 340
320 LET P = P + 1
325 IF P <= N THE 315
330 LET P = 1
335 GOTO 315
340 LET H$(P) = X$
345 NEXT K
400 REM SALIDA DE LA TABLA
405 PRINT "TABLA DE BARAJADO"
407 FOR K = 1 TO N
410 PRINT H$(K)
415 NEXT K
420 END
```

b) Si **J** es el contador

```
230 LET J = 0
312 LET J = J + 1
```

Las líneas 325 y 335 se modifican para bifurcar a la 312

```
417 PRINT "NUMERO DE COMPROBACIONES"; J
```

Ejercicio 16

2.
```
1100 REM EJERCICIO 16 PREGUNTA 2
1102 REM MODULO DE ENTRADA
1105 PRINT "INTRODUCE EL TAMANO DEL CONJUNTO";
1110 INPUT M
1115 IF M <= 25 THEN 300
1120 PRINT "EL LIMITE ES 25"
1125 PRINT "POR FAVOR UTILICE UNA CIFRA MENOR"
1130 GOTO 1105
1150 FOR K = 1 TO M
1155 INPUT N(K)
1160 NEXT K
```

3. *a*) Modificación del programa ejemplo 16.1: se usa la variable **C** para contar el número de comparaciones y se insertan las siguientes líneas:

```
1225 LET C = 0
1245 LET C = C + 1
1435 PRINT "NUMERO DE COMPROBACIONES:"; C
```

Las líneas 1310 y 1420 se transforman como sigue:

```
1310 IF L <= U THEN 1245
1425 GOTO 1435
```

6. *a*) y *b*)

```
500 REM EJERCICIO 16, PREGUNTA 6
505 REM LOCALIZACION DE UN ELEMENTO EN UNA TABLA
        DE BARAJADO
510 REM
515 DIM H$ (25)
600 REM EL SEGMENTO DE ENTRADA O CREACION DE UNA
        TABLA DE BARAJADO
605 REM SE DEBE INSERTAR AQUI, LONGITUD N
610 REM
700 REM INTRODUCCION DEL ELEMENTO QUE VA A SER
        LOCALIZADO
705 PRINT "QUE ELEMENTO VA A SER LOCALIZADO?";
710 INPUT X$
715 REM PUESTA A CERO DEL CONTADOR DE COMPARA
        CION
720 LET C = 0
725 LET P = INT((ASC(X$) - 64) * N/26)
730 LET C = C + 1
735 IF H$(P) = X$ THEN 805
740 IF C > N THEN 815
745 LET P = P + 1
750 IF P <= N THEN 730
755 LET P = 1
760 GOTO 730
800 REM SALIDA
805 PRINT "EL ELEMENTO SE ENCUENTRA EN LA POSI
           CION"; P
810 GOTO 820
815 PRINT "EL ELEMENTO NO ESTA EN LA TABLA"
820 PRINT C; "COMPARACIONES"
825 END
```

Observe que se utiliza el número de comparaciones para determinar si el elemento está o no en la tabla.

Ejercicio 17

2. *a*) **C**: 30000, 40000, 50000, 60000, 99999
b) **C**: 99999
c) **C**: 10000, 30000, 20000, 50000, 99999
d) **C**: 10000, 10001; a partir de aquí el programa fallará, porque intentará leer tras el final de un fichero.

3. Las dos series de entrada contienen datos en orden ascendente a los que pone fin el valor 99999, que no aparece en ningún otro lugar de los mismos.

7. El programa de refundición-ordenación recogido aquí utiliza como subprograma el programa ejemplo 17.1.

```
100 REM EJERCICIO 17, PREGUNTA 7
105 REM PROGRAMA DE ORDENACION-REFUNDICION
110 REM PARA UNA MATRIZ DE 64 ELEMENTOS NUMERI
        COS
115 DIM X(64), A(33), B(33), C(65)
120 REM
200 REM LA ENTRADA DE ELEMENTOS DE X
205 REM DEBE ESCRIBIRSE AQUI
210 REM
300 REM ORDENACION-REFUNDICION
305 FOR P = 1 TO 6
310 REM P: NUMERO DE PASADAS DE REFUNDICION
312 LET N = 2 ^ P
315 FOR Q = 1 TO 64 STEP N
320 REM Q: RECORRER LAS MATRICES DURANTE CADA
           PASADA DE REFUNDICION
325 REM COPIAR A LAS MATRICES A Y B
330 FOR R = 1 TO N/2
335 LET A(R) = X(Q + R - 1)
340 LET B(R) = X(Q + N/2 + R - 1)
345 NEXT R
350 LET A(N/2 + 1) = 99999
355 LET B(N/2 + 1) = 99999
360 REM LLAMADA AL SUBPROGRAMA DE REFUNDICION
365 GOSUB 1000
370 REM COPIAR DE LA MATRIZ C
375 FOR R = 1 TO N
380 LET X(Q + R - 1) = C(R)
385 NEXT R
390 NEXT Q
395 NEXT P
400 REM LA SALIDA DE LOS ELEMENTOS DE X
405 REM SE DEBE ESCRIBIR AQUI
410 REM
1000 REM PROGRAMA EJEMPLO 17.1
1005 REM INCLUIDO AQUI
1010 REM
1210 RETURN
1215 END
```

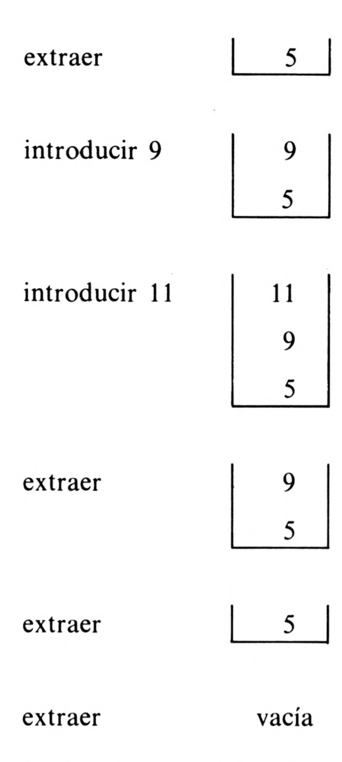
Ejercicio 18

2. introducir 5

5

introducir 7

7
5



3. Si se usa la variable **T** como cima de la pila, la única diferencia entre este subprograma y el de extracción se encuentra en la línea 2210, que se transforma en:

```
2210 REM T: CIMA DE LA PILA
```

y en las líneas 2235 y 2240, sustituidas por:

```
2235 LET T = S(P + 1)
```

Ejercicio 19

2. insertar 3168 — 3168

insertar 4267 — 3168, 4267

eliminar el primer elemento — 4267

insertar 3135 — 4267, 3135

eliminar el primer elemento — 3135

eliminar el primer elemento — vacía

insertar 3258	3258
eliminar el primer elemento	vacía

3.
```
2300 REM EJERCICIO 19, PREGUNTA 3
2305 REM CALCULO DE LA LONGITUD DE UNA COLA
2310 REM PARAMETROS
2315 REM L: LONGITUD DE LA COLA
2320 IF F < I THEN 2335
2325 LET L = F - I
2330 RETURN
2335 LET L = F + 25 - I
2340 RETURN
```

Ejercicio 20

5.
```
2500 REM BORRAR EL PRIMER ELEMENTO DE UNA LISTA
2505 REM PARAMETROS
2510 REM S: INDICADOR DE ACIERTO/FALLO
2515 IF D <> 0 THEN 2530
2520 LET S = 0
2525 RETURN
2530 LET T = L(D + 1)
2535 LET L(D + 1) = F
2540 LET F = D
2545 LET D = T
2550 LET S = 1
2555 RETURN
```

6. La estructura de datos que almacena la cola es una lista encadenada como la descrita en la sección 20.2. con un indicador adicional que señala hacia el último elemento de la cola (variable **R**).

```
1000 REM EJERCICIO 20, PREGUNTA 6
1005 REM MANIPULACION DE COLAS USANDO UNA LISTA
         ENCADENADA
1010 REM VARIABLES GLOBALES
1015 REM Q: MATRIZ PARA ALMACENAR UNA COLA
1020 REM F: INDICADOR INICIAL
1025 REM R: INDICADOR FINAL
1030 REM P: INDICADOR DE LA ZONA LIBRE
1035 DIM Q(30)
```

Crear una cola vacía

```
2000 REM CREACION DE UNA COLA VACIA
2005 REM SIN PARAMETROS
2010 LET F = 0
2015 LET R = 0
2020 LET P = 1
2025 FOR K = 2 TO 28 STEP 2
```

```
2030 LET Q(K) = K + 1
2035 NEXT K
2040 LET Q(30) = 0
2045 RETURN
```

Insertar un elemento en la cola

```
2100 REM INSERTAR UN ELEMENTO EN UNA COLA
2105 REM PARAMETROS
2110 REM A: INSERCION DE ELEMENTO
2115 REM S: INDICADOR DE ACIERTO/FALLO
2120 IF P <> 0 THEN 2135
2125 LET S = 0
2130 RETURN
2135 REM DECIDE SI LA COLA EN CURSO ESTA VACIA
2140 IF R = 0 THEN 2155
2145 LET Q(R + 1) = P
2150 GOTO 2160
2155 LET F = P
2160 LET R = P
2165 LET P = Q(P + 1)
2170 LET Q(R) = A
2175 LET Q(R + 1) = 0
2180 LET S = 1
2185 RETURN
```

Eliminar un elemento de la cola

```
2200 REM ELIMINACION DE UN ELEMENTO DE UNA COLA
2205 REM PARAMETROS
2210 REM B: ELEMENTO EXTRAIDO
2215 REM S: INDICADOR DE ACIERTO/FALLO
2220 IF F <> 0 THEN 2235
2225 LET S = 0
2230 RETURN
2235 LET B = Q(F)
2240 LET T = Q(F + 1)
2245 LET Q(F + 1) = P
2250 LET P = F
2255 LET F = T
2260 REM VERIFICAR SI LA COLA ESTA AHORA VACIA
2265 IF F <> 0 THEN 2275
2270 LET R = 0
2275 LET S = 1
2280 RETURN
```

Ejercicio 21

5.
```
2500 REM ENTREGA UN INDICADOR AL SUBARBOL DERECHO
          DE UN ARBOL DADO
```

```
2505 REM PARAMETROS
2510 REM T: PUNTERO DE ARBOL
2515 REM R: PUNTERO DEL SUBARBOL DERECHO
2520 REM S: INDICADOR DE ACIERTO/FALLO
2525 IF T <> 0 THEN 2540
2530 LET S = 0
2535 RETURN
2540 LET R = VAL(A$(T + 2))
2545 LET S = 1
2550 RETURN
```

8. Observe que el subprograma expuesto aquí es recurrente y que, por tanto, no funcionará en casi ninguna de las versiones de ejecución práctica del Basic.

```
2600 REM RECORRER EL ARBOL
2605 REM PARAMETROS
2610 REM T: PUNTERO DE ARBOL
2615 IF T = 0 THEN 2650
2620 LET T1 = T
2625 LET T = VAL(A$(T1 + 1))
2630 GOSUB 2600
2635 PRINT A$(T1)
2640 LET T = VAL(A$(T1 + 2))
2645 GOSUB 2600
2650 RETURN
```

Ejercicio 22

4.

```
9000 REM EJERCICIO 22, PREGUNTA 4
9005 REM MODULO REVISADO DE VALIDACION
9010 REM PARA PROGRAMA EJEMPLO 22.1
9015 REM PARAMETROS
9020 REM X$: NUMERO DE CUENTA O ELEMENTO
9025 REM V: INDICADOR DE VALIDEZ O NO
9030 IF LEN(X$) = 6 THEN 9045
9035 LET V = 0
9040 RETURN
9045 LET D = 0
9050 FOR I = 1 TO 5
9055 LET D = D + W(I) * VAL(MID$(X$, I, 1))
9060 NEXT I
9065 IF D - INT(D/26) * 26 = ASC(MID$(X$, 6, 1))
       - 65 THEN 9080
9070 LET V = 0
9075 RETURN
9080 LET V = 1
9085 RETURN
```

6. Hay que llevar a cabo dos modificaciones de consideración: una ampliación de la rutina de inicialización que acepte la introduc-

ción del primer número de factura y lo convalide y un módulo de servicio adicional que, cuando reciba un número de factura, entregue el siguiente de la secuencia con la cifra de verificación adecuada. El número de la factura en curso se lleva en la variable **V$** y el siguiente en la **V1$**.

Modificación del módulo de inicialización

Sustituir las líneas 3080 a 3085 por:

```
3080 PRINT "INTRODUZCA PRIMERO EL NUMERO DE FACTU
           RA"
3085 INPUT V1$
3090 LET X$ = V1$
3095 GOSUB 9000
3100 IF V = 1 THEN 2010
3105 PRINT "NUMERO INCORRECTO DE CUENTA"
3110 PRINT "POR FAVOR VERIFIQUELO Y VUELVA A
           INTRODUCIRLO"
3115 GOTO 3085
```

Nuevo módulo de servicio

Este módulo suma a la cuarta cifra del número y resta 3 a la de verificación, sumándole 11 si el resultado es negativo y sustituyendo el valor 10 por *X*.

```
9500 REM INCREMENTO DEL NUMERO DE FACTURA
9505 REM PARAMETRO
9510 REM X$: NUMERO DE FACTURA
9515 LET X = VAL(MID$(X$, 1, 4)) + 1
9520 LET Y = VAL(MID$(X$, 5, 1)) - 3
9525 IF Y >= 0 THEN 9535
9530 LET Y = Y + 11
9535 IF Y < 10 THEN 9550
9540 LET Y$ = "X"
9545 GOTO 9555
9550 LET Y$ = STR$(Y)
9555 LET X$ = STR$(X) + Y$
9560 RETURN
```

Modificación del módulo de extensión de facturas

Insertar las líneas:

```
4001 LET V$ = V1$
4002 PRINT "NUMERO DE FACTURA"; V$
```

Sustituir las líneas 4310 y 4315 por:

```
4310 LET X$ = V$
```

```
4315 GOSUB 9500
4320 LET V1$ = X$
4325 GOTO 2010
4330 REM
```

Modificación del módulo de archivar facturas

Sustituir la línea 7010 por:

```
7010 WRITE #10, V$, D$, A$, N$, R$, K
```

Ejercicio 23

2. *a*) La ampliación a números negativos y fracciones plantea varias dificultades, relacionadas casi todas ellas con la división. Como solución se sugiere la siguiente:

- Ampliar el módulo de inicialización para incluir la entrada y convalidación de un grado de dificultad.
- Cambiar el módulo encargado de generar un número aleatorio comprendido entre 1 y 100 por otro que relacione el intervalo con el grado de dificultad (por ejemplo, 20 veces el grado de dificultad) y cambie el número a una cantidad negativa o a una fracción elegidas al azar utilizando una probabilidad basada en el nivel de dificultad mencionado.
- Corregir el módulo de división para transformar en enteros positivos los números negativos y las fracciones.
- Al final del bucle principal de control, volver a la introducción y convalidación de un nivel de dificultad.

Veremos a continuación el nuevo módulo de generación de números y el de división corregido. La variable **G** representa el nivel de dificultad:

```
500 REM GENERACION DE NUMEROS ALEATORIOS DE
        ACUERDO
505 REM CON UN GRADO DE DIFICULTAD
510 LET X = INT(20 * G * RND(1) + 1)
515 REM CAMBIO A UN NUMERO NEGATIVO AL AZAR
520 IF RND(1) > (G - 1)/20 THEN 530
525 LET X = -X
530 REM CAMBIO A UNA FRACCION AL AZAR
535 IF RND(1) > (G - 1)/20 THEN 545
540 LET X = X/10
545 RETURN
```

Instrucciones adicionales para el subprograma de división

```
902 LET A = INT(ABS(A)) + 1
903 LET B = INT(ABS(B)) + 1
```

Ejercicio 24

3. Los módulos siguientes están escritos en la versión de referencia con el subprograma de escritura de caracteres de la sección 24.5. Incluyen el cruce automático de los límites de la pantalla.

a)
```
9300 REM EJERCICIO 24, PREGUNTA 3A
9305 REM CONTORNO DE UN RECTANGULO
9310 REM PARAMETROS
9315 REM X1, Y1: COORDENADAS DEL VERTICE INFERIOR
                 DERECHO
9320 REM X2, Y2: COORDENADAS DEL VERTICE SUPERIOR
                 DERECHO
9325 REM C$: CARACTER IMPRESO
9330 REM LADO IZQUIERDO
9335 LET X = X1
9340 FOR Y = Y1 TO Y2
9345 GOSUB 9100
9350 NEXT Y
9355 REM LADO SUPERIOR
9360 LET Y = Y2
9365 FOR X = X1 + 1 TO X2
9370 GOSUB 9100
9375 NEXT X
9380 REM LADO DERECHO
9385 LET X = X2
9390 FOR Y = T2 - 1 TO Y1 STEP -1
9395 GOSUB 9100
9400 NEXT Y
9405 REM LADO INFERIOR
9410 LET Y = Y1
9415 FOR X = X2 - 1 TO X1 + 1 STEP -1
9420 GOSUB 9100
9425 NEXT X
9430 RETURN
```

b)
```
9500 REM EJERCICIO 24, PREGUNTA 3B
9505 REM SOMBREAR EL TRIANGULO
9510 REM PARAMETROS
9515 REM X1, Y1: COORDENADAS DEL VERTICE INFERIOR
                 DERECHO
9520 REM X2, Y2: COORDENADAS DEL VERTICE SUPERIOR
                 DERECHO
9525 REM C$: CARACTER UTILIZADO
9530 REM P: PROBABILIDAD DE SOMBREADO
9535 FOR Y = Y1 TO Y2
9540 FOR X = X1 TO X2
```

```
9545 IF RND(1) > P YHEN 9555
9550 GOSUB 9100
9555 NEXT X
9560 NEXT Y
9565 RETURN
```

c)

```
9600 REM EJERCICIO 24, PREGUNTA 3C
9605 REM DIBUJAR EL CONTORNO DE UN CIRCULO
9610 REM PARAMETROS
9615 REM X1, Y1: COORDENADAS DEL CENTRO
9620 REM R: RADIO
9625 REM C$: CARACTER IMPRESO
9630 FOR X = X1 - R TO X1 + R
9635 LET Y = Y1 + SQR(R * R -(X - X1) * (X - X1))
9640 GOSUB 9100
9645 LET Y = Y1 - SQR(R * R -(X - X1) * (X - X1))
9650 GOSUB 9100
9655 NEXT X
9660 RETURN
```

d)

```
9700 REM EJERCICIO 24, PREGUNTA 3D
9705 REM DIBUJO DE UNA LINEA RECTA ENTRE DOS
         PUNTOS
9710 REM PARAMETROS
9715 REM X1, Y1: COORDENADAS DEL PRIMER PUNTO
9720 REM X2, Y2: COORDENADAS DEL SEGUNDO PUNTO
9725 REM C$: CARACTER DIBUJADO
9730 REM DETERMINA LA DIRECCION DE IMPRESION
9735 LET S = SGN(X2 - X1)
9740 FOR X = X1 TO X2 STEP S
9745 LET Y = Y1 + (X - X1) * (Y2 - Y1)/(X2 - X1)
9750 GOSUB 9100
9755 NEXT X
9760 RETURN
```

Ejercicio 25

2. No, porque en algún momento el retraso provocará la entrada de uno de los hechos en la trayectoria crítica, y a partir de ahí todo retraso adicional repercutirá en toda la empresa.

4. *a*) Fecha última = 7.

b)

```
500 REM CALCULO DE LAS ULTIMAS FECHAS
505 LET E(N2, 2) = E(N2, 1): REM HECHO FINAL
510 FOR K = N2 - 1 TO 1 STEP -1: REM RECORRER
    LOS PRIMEROS HECHOS
515 LET E(K, 2) = E(K, 1): REM INICIALIZACION
    DEL ULTIMO SUCESO
520 FOR L = 1 TO N1: REM RECORRER TODAS LAS
    ACTIVIDADES
525 REM RECHAZAR TODAS LAS ACTIVIDADES QUE NO
        COMIENCEN EN EL SUCESO K
```

```
530 IF A(L, 1) <> K THEN 565
535 REM CALCULAR FECHA DEL ULTIMO SUCESO
540 REM SUCESO-DURACION
545 LET T = E(A(L, 2), 2) - A(L, 3)
550 REM COMPARAR CON LA FECHA DEL ULTIMO SUCESO
         ACTUAL
555 IF T >= E(K, 2) THEN 565
560 LET E(K, 2) = T: REM ACTUALIZACION DE
    LA FECHA DEL ULTIMO SUCESO
565 NEXT L
570 NEXT K
```

c) Las líneas 610 y 620 se sustituyen por:

```
610 PRINT "SUCESO"; TAB(10); "PRIMER SUCESO";
612 PRINT TAB(30); "ULTIMO SUCESO"
620 PRINT K; TAB(10); E(K, 1); TAB(30); E(K,2)
```

Ejercicio 26

3. Función para el programa ejemplo 26.1

```
150 REM LA FUNCION PARA LA ECUACION TRASPUESTA
155 REM DE LA LINEA SIGUIENTE SE DEBE EDITAR
        PARA LA FUNCION USADA
160 DEF FNT(X) = (X + 1) ^ 0.25
165 REM
```

Sustituir la línea 320 por:

```
320 LET X2 = FNT(X1)
```

Función para el programa ejemplo 26.3

```
150 REM LA FUNCION QUE VA SER INTEGRADA
155 REM EN LA LINEA SIGUIENTE SE DEBE EDITAR
        PARA USO DE LA FUNCION
160 DEF FNI(X) = (X * X + 1)/(X * X - 1)
165 REM
```

Sustituir las líneas 310 a 320 por:

```
310 LET F1 = FNI(A)
```

Sustituir las líneas 340 y 345 por:

```
340 LET S = S + FNI(X)
```

Sustituir las líneas 355 a 365 por:

```
335 LET F2 = FNI(B)
```

4. Trasposiciones recomendadas:

a) $x = \sqrt[3]{2x}$

b) $x = \sqrt{3 - \frac{1}{x^2}}$

c) $x = e^{1/x}$

d) $x = \log(x + 1)$ (logaritmo natural)

e) $x = \sqrt[4]{3x}$

5. *c*) Derivadas de los miembros izquierdos de las ecuaciones de la pregunta 4:

(*a*) $3x^2 - 2$

(*b*) $2x - \frac{2}{x^3}$

(*c*) $x \log x - x + \frac{1}{x^2}$

(*d*) $e^x - 1$

(*e*) $4x^3 - 3$

7. Suponiendo que la matriz y la matriz unidad se hayan almacenado en una formación de **N** filas y **2N** columnas, el único cambio que debe introducirse en el módulo de creación de una matriz triangular superior es la sustitución de la línea 320 por:

```
320 FOR J = K TO 2 * N
```

El módulo de restitución se encierra en otro bucle que recorre las filas de la matriz inversa **X**. La línea 430 se corrige como se indica a continuación:

```
402 FOR I = 1 TO N
430 LET X(K, I) = (A(K, N + I) - S)/A(K, K)
437 NEXT I
```

10. El módulo de integración del programa ejemplo 26.3 puede volver a escribirse como sigue:

```
300 REM INTEGRACION POR LA REGLA DE SIMPSON
305 LET H = (B - A)/N
310 LET X = A
315 GOSUB 500
320 LET F1 = F
325 LET X = A + H
330 LET S1 = 0
335 FOR K = 1 TO N - 1 STEP 2
340 GOSUB 500
```

```
345 LET S1 = S1 + F
350 LET X = X + 2 * H
355 NEXT K
360 LET X = A + 1 + H
365 LET S2 = 0
370 FOR K = 2 TO N - 2 STEP 2
375 GOSUB 500
380 LET S2 = S2 + F
385 LET X = X + 2 * H
390 NEXT K
395 LET X = B
400 GOSUB 500
405 LET F2 = F
410 LET I = H * (F1 + F2 + 4 * S1 + 2 * S2) / 3
```

Ejercicio 27

2. *a*) Intervalos de valores de los parámetros del modelo:

P Positivo.
L Positivo; puede ser inferior a **P**.
A Positivo pequeño (generalmente comprendido entre 0 y 1).
T2 Positivo.
T1 Positivo comprendido entre **T2**/100 y **T2**/10.

6. Líneas adicionales del segmento de entrada:

```
252 PRINT "NUMERO CAZADO O PESCADO EN CADA INTER
          VALO DE TIEMPO?";
253 INPUT D
```

La línea 530 del segmento de cálculo se sustituye por:

```
530 LET P1 = (A * ( 1 - P/2) * P - D) * T1
```

Ejercicio 28

2.

-14	correcto.
23,6	carácter incorrecto.
47	correcto.
$3 + 42$	signo dentro del entero.

3.

Entrada	Salida
$A * (B + 2)$	$AB2 + *$
$(3 - B) * (2 + A)$	$3B - 2A + *$
$5/(K + 3 + M)$	$5K3 + M + /$

4. *a*) +43.2∇ correcto, salida 2.
1.5∇ correcto, salida 2.
23∇ correcto, salida 1.
−0.4∇ incorrecto, error 2.
−4, 2, 6∇ incorrecto, error 4.

b) error 1: carácter incorrecto.
error 2: número incompleto.
error 3: signo duplicado.
error 4: coma decimal duplicada.
salida 1: entero con signo correcto.
salida 2: decimal con signo correcto.

5. *a*) La forma normal de número es un signo + o − opcional
seguido de
una sola cifra
seguida de
una coma decimal
seguida de
cero o más cifras
seguidas de
la letra E
seguida opcionalmente de
un signo + o −
seguido de
una o más cifras.

b)

Estado	+	−	Cifra		E	Otro carácter	∇
1	2	2	3	error 4	error 5	error 1	error 2
2	error 3	error 3	3	error 4	error 5	error 1	error 2
3	error 3	error 3	error 6	4	error 5	error 1	error 2
4	error 3	error 3	4	error 4	5	error 1	error 2
5	6	6	7	error 4	error 5	error 1	error 2
6	error 3	error 3	7	error 4	error 5	error 1	error 2
7	error 3	error 3	7	error 4	error 5	error 1	salida

Ejercicio 29

3. En el lenguaje ensamblador del MOS:

a)

Etiqueta	*Instrucción*	*Interpretación*
	CLR	Borrar el acumulador.
	ALM T	Almacenar cero en el total.

N	**INT**	Introducir un número.
	BZE A	Bifurcar a la etiqueta **A** si el número es cero.
	SUM T	Sumar el total al número.
	ALM T	Almacenar el nuevo total.
	BRN N	Bifurcar para introducir el siguiente número.
A	**CAR T**	Cargar el total en el acumulador.
	SAL	Llevar el total a la salida.
	DET	Parar.
T	**DTA**	Almacenamiento para el total.
	FIN	Fin del programa.

b)

	INT	Introducir el primer número.
	ALM A	Almacenar el primer número.
	INP	Introducir el segundo número.
	ALM B	Almacenar el segundo número.
	NEG	Negativizar el segundo número. todavía en el acumulador.
	INC	Incrementar el acumulador (se obtiene así el complemento negativo del segundo número).
	SUM A	Sumar el primer número.
	BNG C	Bifurcar si es negativo (el segundo número es mayor).
	CAR A	Cargar el primer número.
	BRN D	Bifurcar a la salida.
C	**CAR B**	Cargar el segundo número.
D	**SAL**	Llevar a la salida el número mayor.
	DET	Parar.
A	**DTA**	Almacenamiento para el primer número.
B	**DTA**	Almacenamiento para el segundo número.
	FIN	Fin del programa.

Indice alfabético

ANAYA MULTIMEDIA

Colección «MICROINFORMATICA»

PROGRAMACION AVANZADA EN BASIC.—Bishop, P.
PASCAL A PARTIR DEL BASIC.—Brown, P.
EL ORDENADOR PERSONAL: COMO ELEGIRLO Y UTILIZARLO.—Cavalcoli, A.
PROGRAMACION EN BASIC: UN METODO PRACTICO.—Dachslager, H.; Hayashi, M.; Zucker, R.
TU PRIMER LIBRO DEL ZX SPECTRUM.—Dewhirst, J.; Tennison, R.
ASTRONOMIA: EL UNIVERSO EN TU ORDENADOR (ZX Spectrum).—Gavin, M.
EL ORDENADOR Y TUS HIJOS.—Hammond, R.
EL LIBRO GIGANTE DE LOS JUEGOS PARA ORDENADOR.—Hartnell, T.
BITS Y BYTES: INICIACION A LA INFORMATICA.—Heller, R. S.; Martin, C. D.
MICROINFORMATICA. Conceptos básicos.—Hollerbach, L.
DESCUBRE LAS MATEMATICAS CON TU MICRO.—Johnson, D.
EL ORDENADOR EN EL AULA.—Pentiraro, E.
EL LIBRO DEL BASIC.—Zaks, R.

DISEÑO DE GRAFICOS Y VIDEOJUEGOS (ZX Spectrum).—Angell, I. O.; Jones, B. Y.
MATEMATICAS DIVERTIDAS EN BASIC.—Kosniowski, C.
INTELIGENCIA ARTIFICIAL. CONCEPTOS Y PROGRAMAS.—Hartnell, T.
¿QUE ES LA TELEMATICA? Nuevas tecnologías en la sociedad de la información.—Servello, F.
COMO SE PROGRAMAN LOS ORDENADORES. Programación estructurada básica.—De Rosso, V.
«SPRITES» Y GRAFICOS EN LENGUAJE MAQUINA (ZX Spectrum).—Durst, J.
EL LIBRO GIGANTE DE LOS JUEGOS PARA ZX SPECTRUM.—Hartnell, T.
LENGUAJE MAQUINA AVANZADO (ZX Spectrum).—Webb, D.
LOS ORDENADORES NO MUERDEN.—Coccione, L.; Winter, G.

El propósito de este libro es enseñar a programar en un lenguaje de alto nivel, utilizando el BASIC como modelo para presentar las técnicas de programación y desarrollar los algoritmos.

PROGRAMACION AVANZADA EN BASIC responde a las necesidades de la industria informática, de los estudiantes de programación de ordenadores y de los poseedores de un microordenador que no se conformen con un conocimiento superficial del BASIC y deseen mejorar su capacidad de diseño de programas y desarrollo de aplicaciones.

A lo largo de sus cinco apartados se analizan de forma exhaustiva todos los aspectos de la programación: fundamentos, programación estructurada y diagramas de flujo; diseño, estructura, verificación y mantenimiento de programas; operaciones fundamentales de programación; pilas, colas, listas y estructuras de datos. La última parte del libro se dedica a aplicaciones, como los gráficos, la simulación de procesos y el análisis sintáctico.

PROGRAMACION AVANZADA EN BASIC es el texto definitivo de programación de ordenadores, al reunir en un acertado planteamiento didáctico las técnicas más útiles para el diseño y desarrollo de programas de alta calidad.




Programación avanzada
en BASIC
Peter Bishop


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/